Design of Object Detection Systems on a SoC-FPGA Platform

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines

Doktoringenieurs (Dr.-Ing.)

genehmigte Dissertation

von

Shaodong Qin

geboren am 20.10.1988

in Shandong, China

Eingereicht am: 30.08.2017
Disputation am: 13.11.2017

1. Referent: Prof. Dr.-Ing. Mladen Berekovic
Technische Universität Carolo-Wilhelmina zu Braunschweig
2. Referent: Prof. Dr.-Ing. Harald Michalik
Technische Universität Carolo-Wilhelmina zu Braunschweig

2017

Abstract

Object detection is a crucial and challenging task in the field of embedded vision and robotics. Over the last 15 years, various object detection algorithms/systems (e.g., face detection, traffic sign detection) are proposed by different researchers and companies. Most of these research are either focused on improving the detection performance or devoted to boosting the detection speed, which reveals two typically employed criteria while evaluating an object detection algorithm/system: detection accuracy and execution speed. Considering these two factors and the application of object detection to the domains such as service robots and Advanced Driving Assistance System (ADAS), FPGA is a promising platform to achieve accurate object detection in real-time. Therefore, FPGA-based robust object detection systems are designed in this work.

The main work of this thesis can be divided into two parts: promising algorithm obtaining and hardware design on SoC-FPGA. Firstly, representative object detection algorithms are selected, implemented, and evaluated. Thereafter, a generalized object detection framework is created. With this framework, pedestrian detection, traffic sign detection, and head detection algorithms are realized and tested. The experiments verify that promising detection results can be obtained by employing the generalized object detection framework. For the work of hardware design on FPGA, the platform of the object detection system, which consists of stereo OV7670 cameras, Xilinx Zedboard, and a monitor that can visualize the detection results, is created. After that, IP cores that correspond to each block of the framework are designed. Configurable parameters are provided by each IP core so that the IPs, especially feature calculation IP and feature scaler IP, can be correctly instanced according to the fast feature pyramid theory. Finally, by employing the designed IP cores, pedestrian detection system, traffic sign detection system, and head detection system are designed and evaluated. The on-board testing results show that real-time object (e.g., pedestrian, traffic sign, head) detection with promising accuracy can all be achieved. In addition, with the generalized object detection framework and the designed IP-toolbox, the object detection system that targets any instance of objects can be designed and implemented rapidly.

Kurzfassung

Objekterkennung ist eine essenzielle und herausfordernde Aufgabe in den Forschungsgebieten der Embedded Vision und der Robotik. In den letzten 15 Jahren wurden verschiedene Objekterkennungsalgorithmen und Systeme (z.B. Gesichtserkennung) aus der Forschung sowie der Industrie präsentiert. Der Forschungsschwerpunkt liegt dabei typischerweise entweder bei der Verbesserung der Erkennungsqualität oder bei der Beschleunigung der Erkennungsgeschwindigkeit. Hieraus leiten sich direkt die Kriterien für den Einsatz von Objekterkennungsalgorithmen und Systemen ab: Erkennungsgenauigkeit und die Erkennungslatenz. Unter Berücksichtigung dieser Kriterien und dem konkreten Einsatzgebiet der Objekterkennung für Serviceroboter und ADAS haben sich FPGA Plattformen als vielversprechende Kandidaten für die Implementierung von hoch genauen Objekterkennungsalgorithmen mit strikten Echtzeitanforderungen herausgestellt. Auf Bases dessen wurden im Rahmen dieser Arbeit eine robuste Objekterkennung entwickelt.

Der Fokus dieser Arbeit ist geteilt in zwei Aspekte: Identifikation und Analyse geeigneter Objekterkennungsalgorithmen und deren Implementierung in einer FPGA basierten SoC-Plattform. Zu Beginn wurden eine Reihe von repräsentativen Algorithmen ausgewählt, in einer Testumgebung implementiert und evaluiert. Darauf aufbauend wurde ein Entwicklungs-Frameworks für die Erkennung von Passanten, Verkehrszeichen sowie Kopfdetektion entwickelt und analysiert. Für die Entwicklung einer FPGA basierten Plattform wurde ein Objekterkennungssystem erstellt. Eine Sammlung aus IP-Cores, die das Entwicklungs-Framework bilden, wurden für die FPGA Plattform implementiert. Die IP-Cores bieten eine Reihe von Konfigurationsparametern für den flexiblen Einsatz von neuen Komponenten, im Besonderen IP-Cores für die Berechnung und Skalierung von Erkennungseigenschaften, basierend auf der Fast Feature Pyramid Theorie. Abschliessend wurden die entwickelten IP-Cores zur Erkennung von Passanten, Verkehrszeichen sowie die Kopfdetektion integriert und gemeinsam evaluiert. Die gesammelten Ergebnisse aus verschiedenen Testscenarios der entwickelten FPGA-Plattform zeigen, dass die Objekterkennung in Echtzeit mit vielversprechender Genauigkeit erreichbar ist. Darüber hinaus können mit dem generalisierten Objekterkennungsrahmen und der entwickelten IP-Toolbox schnell und flexibel belibige Objekterkennungssysteme entwickelt und implementiert werden.

Table of Contents

List of Figures

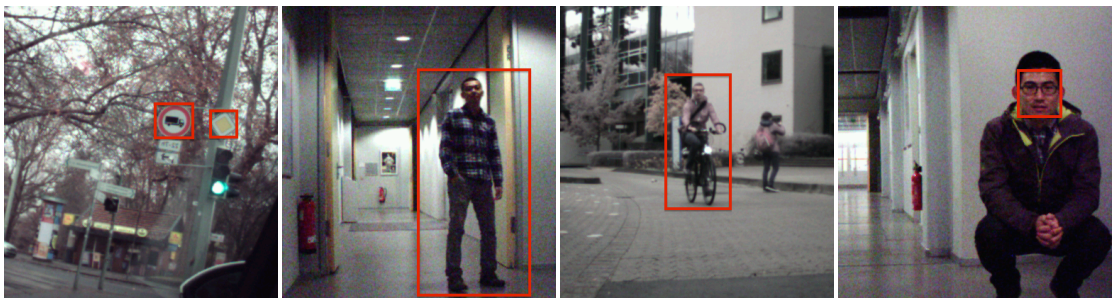
List of Tables

Chapter 1

Introduction

1.1 Background Introduction

Object detection is a hot research topic over the last two decades. The goal of object detection is to detect all instances of objects from a known class, such as cars, traffic signs or people in an image [1]. Today, it is widely used in the field of robotics, computer vision, autonomous driving, etc. Due to the difference of application scenarios, many interesting and useful detection algorithms/applications are proposed and implemented by different researchers or companies. Some of these object detection applications are shown in Figure 1.1.



(a) (b) (c) (d)

Figure 1.1: An illustration of various object detection applications: (a) Traffic sign detection (b) Pedestrian detection (c) Cyclist detection (d) Face detection

Figure 1.1 (a) illustrates an example of traffic sign detection, which is an essential and important part of any traffic sign recognition system. Figure 1.1 (b) (c) (d) illustrate the example applications of pedestrian detection, cyclist detection, and face detection, respectively.

With the popularization of Advanced Driver Assistance Systems (ADAS), different driving-safety modules are added to the ADAS system gradually, which includes several object detection modules, such as front-view car detection, pedestrian detection, and blind spot detection. Today, these object detection modules are be-

coming the essential components of different ADAS or intelligent driving systems. By employing object detection applications in ADAS, the probability of rear-end collision or other car accidents can be dramatically decreased. In addition, traffic sign detection, obstacle detection, etc. applications are also an indispensable part of the autonomous/automated driving system, which is also an extremely popular research field and being developed independently by Google [2], Daimler [3], Continental [4], etc.

Face detection and iris detection are two typical applications in computer vision and robotics field, which normally serve as the first step of facial and iris recognition, respectively. Nowadays, some modern smartphones and laptops employ face or iris detection/recognition as one feature to realize device unlocking or even bill-paying. In medical diagnosis area, suspicious tissues which are similar to breast tumor can be detected by a detection application that employing AdaBoost classifier with Haar-like features [5]. With this application, the doctor can focus more on the suspicious areas that detected and marked by the software.

Object detection is also applied in many other fields, such as surveillance control, smart home, and robotics. In a nutshell, all the works that related to object detection during the last two decades can be categorized into two groups:

- Proposing new detection approaches with better performance. These works are usually achieved by employing better feature descriptors or better classification methods. The targeting platform of these works is PC, and C/C++/ Matlab/python is used as the programming language.
- Improving the detection speed of various object detection algorithms. Unlike new detection methods proposing, this category of works relies on the algorithms proposed by other researchers and devotes the effort to improving the detection speed on different heterogeneous platforms (e.g., CPU+GPU, FPGA). During this process, code optimization for hardware platforms, e.g., CPU+GPU, or delicate hardware design for FPGA is employed.

Take pedestrian detection application as the example, in [6] [7] [8] [9], different pedestrian detection algorithms are proposed to achieve better detection results by utilizing larger feature descriptors (e.g., gradient, LUV, optical flow) or more complex classification methods, such as Convolutional Neural Networks (CNN). These algorithms indeed achieve better detection performance, with the price of requiring more computing resources and slower detection speed. For instance, Tome proposed a pedestrian detection approach based on deep convolutional neural networks in [8], which produces the sate-of-art performance for pedestrian detection. However, the execution time of this application on a machine with six-

core 2.4 GHz Intel Xeon CPU E5-2609, a NVIDIA GTX980 GPU and 32GB ram is 530ms for a 640×480 image.

On the other hand, a number of real-time pedestrian detection systems are implemented on FPGA or SoC-FPGA platform [10] [11] [12] [13]. Although most of the implementations are based on the conventional HoG+SVM [14] algorithm proposed by Dalal in 2005, low-power consumption and real-time pedestrian detection are realized. (A more concrete and detailed description of related works to object detection are introduced in Section 2.4.)

1.2 Contributions

Due to the differences of objects, one object detection algorithm is ordinarily targeting one type of objects. For instance, face detection algorithm is only targeting faces. Consequently, the designed hardware system on FPGA or SoC-FPGA can also only detect one type of objects. The aim of this work is to design a generalized object detection system, which can be used to detect different objects with minimum efforts of reconfiguring some parameters. To achieve this goal, the main work of this thesis is summarized and shown in Figure 1.2.

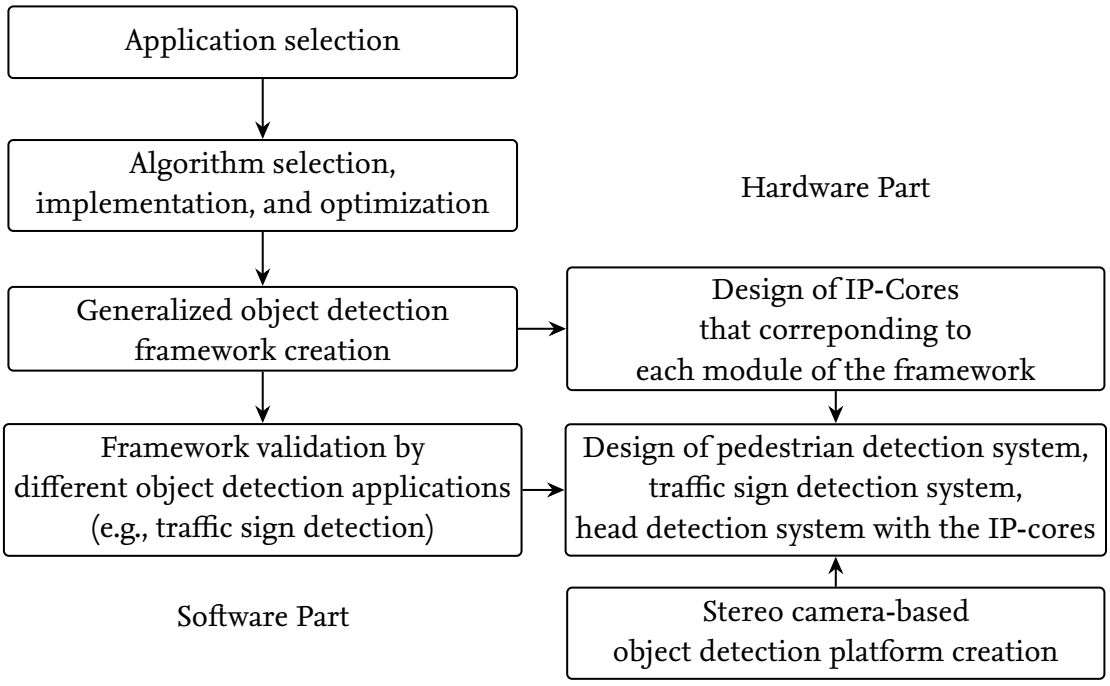


Figure 1.2: The main work of this thesis

- Firstly, the typical and state-of-art object detection algorithms, especially pedestrian detection algorithms (since pedestrian detection is one of the

most difficult tasks in object detection), are selected, implemented, optimized, and analyzed.

- Based on the obtained pedestrian detection algorithm, which achieves one of the best performance, a generalized object detection framework is created and verified by various object detection applications. Then, with this framework, the algorithms of traffic sign detection, head detection, and pedestrian detection are programmed and evaluated.
- A object detection platform, which consists of two OV7670 cameras, Zedboard, and a monitor, is created. To connect stereo cameras with Zedboard via Pmod interface, two PCB boards are designed.
- An IP-toolbox, which includes all IP cores (e.g., feature calculation, classification, camera capture) required to create various object detection systems on SoC-FPGA, is designed. To guarantee the IP cores can handle the detection of various objects, configurable parameters are supported by most of the IP cores.
- Pedestrian detection system, traffic sign detection system, and head detection system are designed designed by employing the IP cores that inside the IP-toolbox. In this process, parameters of each IP core are configured according to the fast feature calculation theory and the trained detector of each system.

1.3 Structure of the Thesis

The structure of this thesis is organized as follows.

Chapter 2 firstly introduces the conventional process of object detection algorithms and systems. Then, typical features and classification methods used in various object detection applications are described. After that, related work of object detection algorithms and systems are introduced and compared.

In Chapter 3, typical and state-of-art object detection algorithms are selected and implemented. The performance of selected algorithms are compared and analyzed.

By analyzing the detection process and performance of selected algorithms, a generalized object detection framework, which provides promising detection results and is suitable for hardware pipeline architecture generation, is created and verified in Chapter 4.

In Chapter 5, based on the generalized object detection framework, all IP cores, such as stereo image rectification, feature extraction, data scaling and detection, are designed by Verilog or Vivado High-Level Synthesis (HLS) tool [15].

Thereafter, in Chapter 6, pedestrian detection system, traffic sign detection system, and head detection system are designed and implemented with the IP cores that designed in Chapter 5. The testing result of each detection system on Zedboard is evaluated.

Finally, Chapter 7 concludes the whole work.

Chapter 2

Object Detection Algorithms and Systems

In this chapter, the typical structure of object detection algorithms and systems is described. Then, the key features and classification methods that used in different detection algorithms are introduced. Based on these pieces of knowledge, related work of object detection algorithms and systems are described.

2.1 Typical Object Detection Structure

After the research of object detection for two decades, various detection algorithms are proposed, and many detection systems are built. However, the overall structure of these algorithms and systems are similar. In this section, the typical structure of object detection is presented. Moreover, as mentioned in Chapter 1.1, the research works in object detection field are either focused on improving the detection accuracy or devoting to accelerating the detection speed on different platforms. Due to the difference of their focus, the typical structure of object detection algorithm and system is separately introduced.

2.1.1 The Typical Structure of Object Detection Algorithm

The targeting objects of an object detection algorithm can be humans, eyes, tennis ball, etc. Because the complexity of these objects is quite different, different classification approaches are applied. For instance, to detect a green tennis ball in an image, the size, shape, and color information of the ball is known. With these bits of information, the tennis ball can be defined by some mathematic restrictions, such as the ball is round, the color is green. Therefore, the detection can be performed by employing these mathematic constraints.

However, for the human detection, face detection, etc., it is impossible to describe the targeting objects by simple mathematics. Hence, employing machine learning approaches in object detection applications emerged and popularized. The machine learning methods that commonly used in object classification are Support

Vector Machine (SVM) [16] and Boosting (e.g., AdaBoost [17]). In recent two years, deep convolutional neural networks [18] based classification approach is also becoming a popular in object detection field.

Figure 2.1 shows the typical structure of an object detection algorithm using machine learning approach.

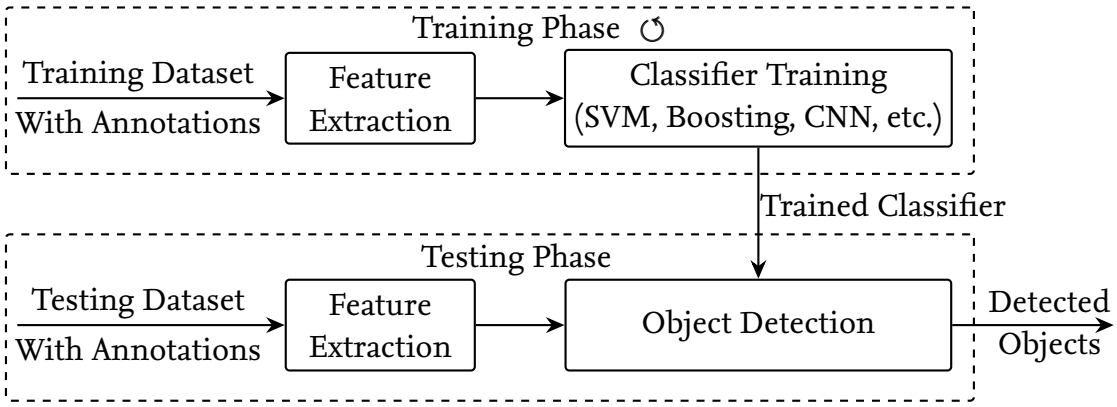


Figure 2.1: The structure of a typical object detection algorithm

The whole algorithm is divided into two phases: training and testing. In training phase, thousands of object images with annotations are used as the training dataset to train the classifier by the classification method SVM, Boosting or CNN. Note that the training phase may repeat multiple times to generate a promising classifier. When the classifier is generated, it is employed in the testing phase to detect whether objects exist in testing images. The images used in testing phase are also annotated. Hence, the performance of the algorithm can be verified and measured.

2.1.2 The Typical Structure of Object Detection System

Except proposing new algorithms, some researchers work on the algorithm optimization or algorithm implementation on different platforms, e.g., PC+GPU, FPGA, embedded systems. For these works, the training phase is omitted, and the pre-trained classifier is directly used. The typical structure of an object detection system is shown in Figure 2.2.

As can be seen from Figure 2.2, only one phase is inside the detection system: detecting phase. The data input of the system can be from an image sensor, such as video camera, infrared camera, or from image dataset. When the targeting plat-

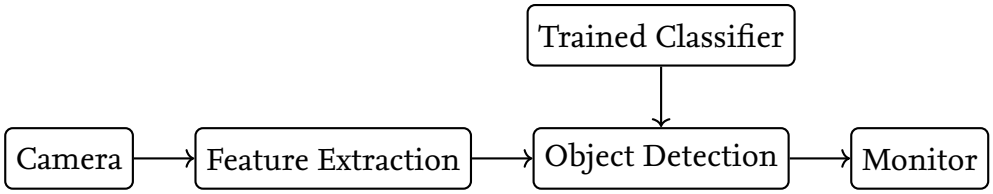


Figure 2.2: The structure of a typical object detection system

form is embedded systems or FPGA, a monitor is also required to display the results of detection.

2.2 Commonly Used Features in Object Detection

For the ease of description, "channel" is used in this thesis to describe the resulting "image" of extracting one type of feature from an image. In other words, a channel is the response of a linear or nonlinear transformation of an input image [19]. Each pixel in a channel is corresponding to one pixel or a patch of pixels in the input image. For instance, R, G, and B color channels of a color image can sever as three channels. Given two images I and I' that are related by a translation, if we denote the calculation of a channel as Ω, two calculated channels $C = \Omega(I)$ and $C' = \Omega(I')$, which are translationally invariant, are related by the same translation.

2.2.1 LUV Feature

Color is a salient feature of the object in many object detection applications, e.g., license plate detection, traffic sign detection. In real applications, the captured colorful images are stored in RGB format. RGB is a widely used color space in the field of photography, computer graphics, and television. It utilizes the combination of red, green, and blue to generate different colors [20]. Hence, the change of color may result in an obvious numerical difference of all three color channels, which makes RGB color space not appropriate for the comparison of image differences.

On the other hand, CIE LUV color space, which is the extension of "CIE 1931 XYZ" color space, was designed to be "perceptually linear". This means that a change of the same amount in a color value should produce a change of about the same visual importance [20]. Because of its "perceptually linear" characteristic, CIE LUV color space is much more suitable for color comparison, difference comparison of color images, etc. tasks. Therefore, L, U, and V are employed as the representing features in some object detection applications.

To use the LUV features in object detection, a conversion from RGB to LUV color space is required. The three channels that comprise the CIE LUV color space are luminance, which is denoted as L and color values that can be represented by U and V. Because CIE LUV is a derivation from standard CIE XYZ color space, no straightforward converting equations are available from RGB to CIE LUV. Hence, RGB color space to CIE XYZ color space is firstly converted, which can be expressed by:

$$\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} 0.430574 & 0.341550 & 0.178325 \\ 0.222015 & 0.706655 & 0.071330 \\ 0.020183 & 0.129553 & 0.939180 \end{bmatrix} \begin{bmatrix} R_i \\ G_i \\ B_i \end{bmatrix}, \tag{2.1}$$

where R_i, G_i and B_i are the R, G and B color value of one pixel, respectively. X_i, Y_i and Z_i are the channel values after conversion. With X_i, Y_i and Z_i, the luminance L_i can be calculated by:

$$L_i = \begin{cases} 116 \cdot (Y_i)^{\frac{1}{3}} - 16, Y_i > (\frac{6}{29})^3 \\ \\ (\frac{29}{3})^3 \cdot Y_i, Y_i \leq (\frac{6}{29})^3 \end{cases} \tag{2.2}$$

Thereafter, the data of U and V channels U_i and V_i can be obtained by

$$\begin{aligned} U_i &= 13L_i(\frac{4X_i}{X_i + 15Y_i + 3Z_i} - 0.197833) \\ V_i &= 13L_i(\frac{9Y_i}{X_i + 15Y_i + 3Z_i} - 0.468331). \end{aligned} \tag{2.3}$$

The conversion matrix employed in Equation (2.1) and parameters used in Equation (2.2), (2.3) are predefined by the CIE organization [21]. With Equations (2.1) to (2.3), the color space conversion from RGB to LUV can be achieved.

Figure 2.3 illustrates the difference between RGB channels and LUV channels of the average image of all positive samples in INRIA dataset [22]. From this figure, it is evident that the intensity of RGB color channels barely changes. On the contrary, obvious differences exist in LUV channels. The red rectangle in the U channel highlights that there is a peak at the location of human's face, which can be used as a key feature in pedestrian detection.

2.2.2 Gradient Feature

The gradient of a pixel in an image is defined as the directional change in each color channel of the pixel relative to its surrounding pixels. For each pixel, its

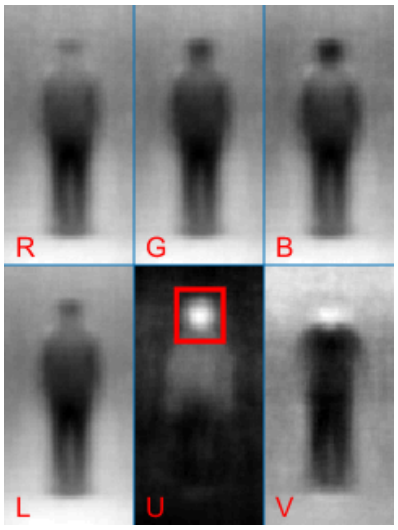


Figure 2.3: The difference between RGB and LUV channels of the average image of all positive samples in INRIA dataset

gradient contains two parts: magnitude and orientation. The gradient magnitude represents the intensity of the directional change, and the gradient orientation represents the direction of this change [23]. In image processing, there are several methods to compute the gradient of an image. The most commonly used is the first-order gradient. The calculation process of first-order gradient can be described as follows.

For any point $P(x,y)$, its surrounding points are $P(x-1,y)$, $P(x,y-1)$, $P(x+1,y)$, and $P(x,y+1)$. The gradient magnitude $M(x,y)$ of pixel $P(x,y)$, which can be decomposed to gradient magnitude along x-axis $M_{dx}(x,y)$ and y-axis $M_{dy}(x,y)$, is calculated by:

$$\begin{aligned} M_{dx}(x,y) &= \frac{P(x+1,y)-P(x-1,y)}{2} \\ M_{dy}(x,y) &= \frac{P(x,y+1)-P(x,y-1)}{2} \\ M(x,y) &= \sqrt{M_{dx}(x,y)^2 + M_{dy}(x,y)^2}. \end{aligned} \tag{2.4}$$

From Equation (2.4), we can obtain that the components $M_{dx}(x,y)$ and $M_{dy}(x,y)$ of gradient magnitude can be represented by the convolution of the image with filters $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$, respectively. Furthermore, with the calculated $M_{dx}(x,y)$ and $M_{dy}(x,y)$, the gradient orientation $O(x,y)$ at point $P(x,y)$ can be obtained

by:

$$O(x,y) = atan(\frac{M_{dy}(x,y)}{M_{dx}(x,y)}). \tag{2.5}$$

Figure 2.4 exemplifies an image captured by OV7670 camera (the camera that employed in the object detection platform of this work, which will be introduced in Section 6.1) and its corresponding gradient magnitude result.

Figure 2.4: An example of the OV7670 camera captured image (left) and its corresponding channel image of the gradient magnitude (right)

As can be seen from Figure 2.4, the contour of objects can be extracted with gradient computing. Therefore, gradient magnitude can be employed as one channel feature for traffic sign, etc. objects which have a clear contour.

2.2.3 HoG Feature

HoG feature, which is the abbreviation of Histogram of Oriented Gradients, was first introduced by Dalal and Triggs for pedestrian detection in [14]. The basic idea of HoG feature is trying to utilize the gradient orientation information that computed during the gradient computing process. Although by Equation (2.5), gradient orientation of all pixels in the image can be calculated, the orientation results are sensitive to even slightly changes in the original image. Therefore, in HoG feature, a histogram is performed for every $N \times N$ data of orientation channel where N is normally between $3 \sim 9$.

Figure 2.5 illustrates an example of a 3×3-based HoG calculation process. Gradient orientation of each pixel is drawn as an arrow and all orientation data are categorized into four bins/groups. To simplify the calculation process and description, assuming the magnitude of all gradients are the same and equal to one. Then, the histogram is executed by accumulating the magnitude of gradient in each bin based on their orientations. Finally, the HoG result can be obtained by

combining the histogram results to one vector, which equals to [4 1 2 2] in the example of Figure 2.5.

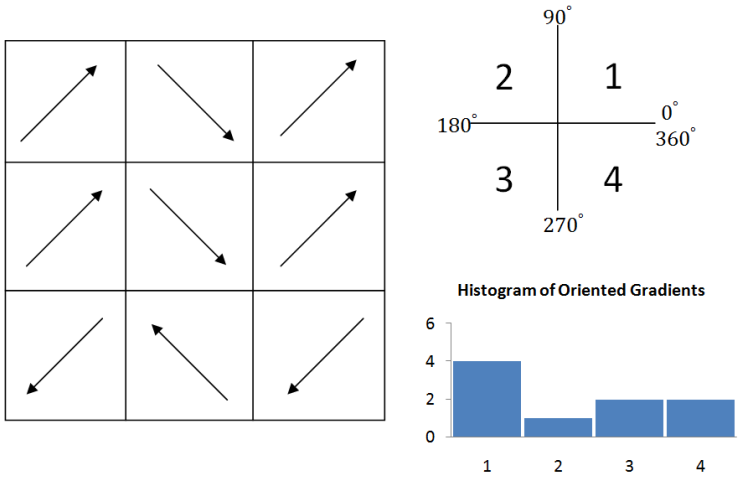


Figure 2.5: An illustration of a simplified HoG calculation process on a 3×3 block data

In original HoG descriptor, a patch of 8×8 pixels is grouped as a cell. 9 orientation bins are defined by evenly spacing $0^o \sim 180^o$ (unsigned gradients). Then, histogram of oriented gradients is calculated by quantizing all orientation data inside each cell into 9 bins. Due to the difference of illuminations, object poses, and foreground-background contrast, an effective local normalization is required for the obtained HoG channel image. This local normalization is separately performed for each block of 2×2 cells which consist of 16×16 pixels. After histogram operation, for a 640×480 image, the dimension of resulting HoG channel image is changed to 160×120.

From the computation process, we can notice that, for gradient feature, only gradient magnitude is used as the feature. And in HoG feature, the gradient orientations are quantized and also used as the features. Because the gradient orientations are classified into 9 bins, 9 channel images are produced by HoG calculation.

With the success of using HoG feature in human detection, it is adapted to a variety of other object detection tasks, such as car detection and traffic sign detection. Moreover, HoG feature with different numbers of bins and different sizes of cells are also tested, compared, and employed by some algorithms. Figure 2.6 exemplifies a HoG-like feature result that generated by ICF [19] algorithm. In this algorithm, the orientation data are summarized to 6 bins. The gradient magnitude and HoG channels of the average image of all positive training samples in INRIA dataset [22] are shown in Figure 2.6. As marked by red arrows in this figure, characteristic contours of body parts like shoulder and legs are highlighted

by different HoG channels. All these characteristic data can be used as the critical features in object detection.

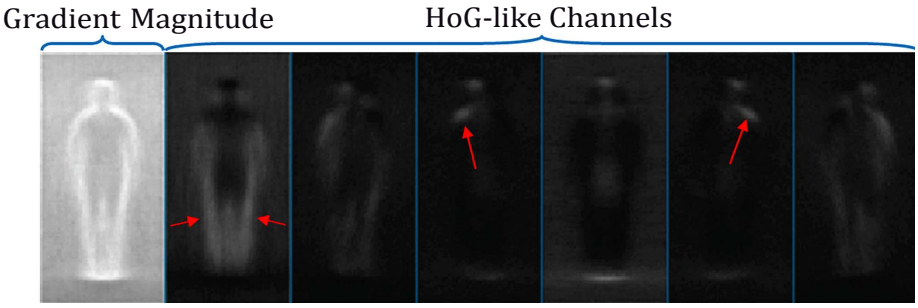


Figure 2.6: The gradient magnitude and HoG-like channels of the average image of all positive training samples in INRIA dataset

2.2.4 Haar-like Feature

As shown in Figure 2.3 and 2.6, characteristics of the pedestrian are handcrafted in different channels. In order to indicate the existence of certain characteristic in particular area of the particular channel, the rectangular Haar-like feature is employed. The haar-like feature is firstly introduced by Viola and Jones [24] in the task of face detection. In their work, a 2-rectangle and a 3-rectangle Haar-like feature are employed to dig out the characteristic that eye region is often darker than the cheeks, as illustrated in Figure 2.7.

Figure 2.7: An illustration of Haar-like features emoloyed in face detection

As a matter of fact, the haar-like feature can also be employed in other object detection applications such as pedestrian detection and traffic sign detection. For instance, in the ICF algorithm [19], 1-rectangle Haar-like feature is used to capture the handcrafted features in each channel. However, since a huge number of computing resources are required to realize Haar-like feature computation, a fast computing method "integral image" is proposed by Viola [25].

As illustrated in Figure 2.8 (left), if we denote the integral image as I, the value of any point (i, j) in I can be calculated by summarizing the pixels that above and to

the left of $I(i,j)$:

$$I_i(i,j) = \sum_{x \le i, y \le j} I(x,y).$$

After the integral image is obtained, the calculation of Haar-like feature value in a random location of image I can be computed by

$$S = I_i(D) - I_i(C) - I_i(B) + I_i(A),$$

where A, B, C, and D are the corner points of the Haar-like feature rectangle, as shown in Figure 2.7 (right).

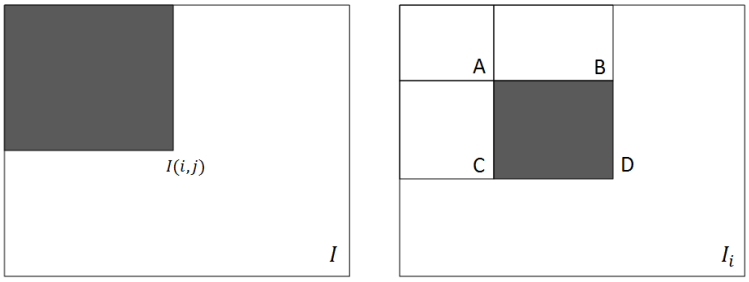


Figure 2.8: The calculation process of Haar-like feature by employing integral image

2.3 Typical Classification Methods in Object Detection

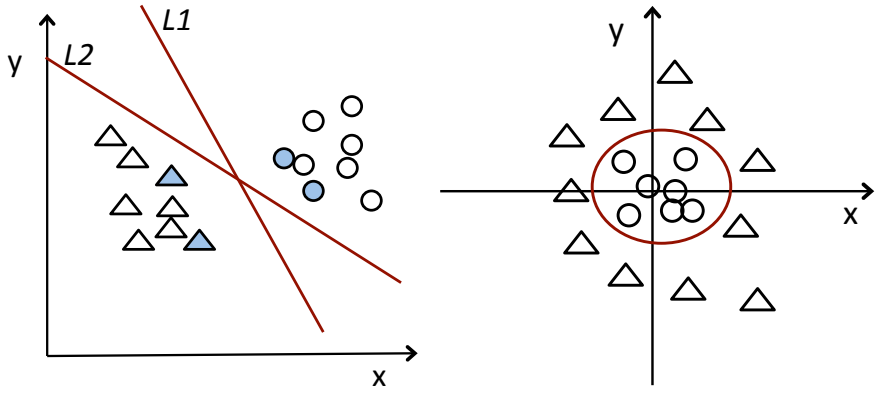
As shown in Figure 2.1, after the features are extracted from training images, machine learning approach can be selected and employed to build the classification function. The classifiers that used in object detection applications mainly include Support Vector Machine (SVM), Boosting, and Convolutional Neural Networks (CNN). In this section, the typically used classification approaches are introduced.

2.3.1 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning approach that mainly used for binary classification. It is one of the most widely used classification approaches in object detection algorithms.

Figure 2.9 gives a binary classification example of classifying the triangles and circles, such as the red lines L1 and L2 in Figure 2.9(a). In SVM, the best classification is defined as the line that can maximize the margin between nearest data points

from both groups and the red line [26]. Therefore, in Figure 2.9(a), the red line L1 is a better classification than the line L2. The nearest data points, marked as blue in Figure 2.9(a), are called supporting vectors, which are the critical features to set the red line up.



(a) Linear-SVM (b) Kernel-SVM

Figure 2.9: An illustration of linear-SVM and kernel-SVM classification methods

For many binary classification tasks, which are not too complex, Figure 2.9(a) illustrated linear-SVM approach can achieve good classification results. Although there are several data points cannot be correctly classified, minimizing the error of false classifications and maximizing the margin distance is still an effective solution. For instance, [14], [27], and [28] all implement human detection with the linear-SVM trained classifier. In [29], a grass region detector is trained by linear-SVM. Moreover, car detection is realized also using the linear-SVM based classifier in [30] and [31].

For the tasks that cannot be easily classified by linear-SVM, as shown in Figure 2.9(b), kernel-SVM can be utilized. Nevertheless, although kernel-SVM can handle more difficult classification problems, it is not true that kernel-SVM is always better than linear-SVM. For many applications, similar detection results are achieved by both kernel-SVM and linear-SVM classifiers. However, the training speed of kernel-SVM is much slowly.

It is worth mentioning that Figure 2.9 is only an example of classification task with two dimensions (/features). In real-life applications, the features that employed for classification are hundreds or thousands. However, the principle concept of SVM that to find the hyperplane by maximizing the margin of all feature dimensions is the same.

2.3.2 Decision Tree

Decision tree is a classification approach, which is the most similar to the human thinking. For a human being, to recognize if an animal is mammals or non-mammals, the might be thinking process is illustrated in Figure 2.10. When the answers to two questions are 'yes', a human will classify this animal to mammals, otherwise, non-mammals [32]. Decision tree based classification employs a similar mind behind it. Each conditional statement is a feature for classification and multiple layers of conditional judgments are processed to get the classification result.

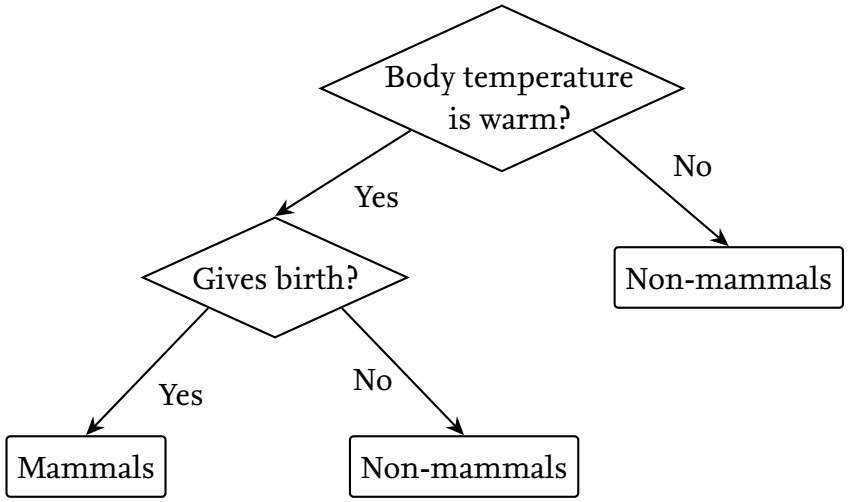


Figure 2.10: An illustration of decision tree-based classification method

The top layer of a decision tree is called root, the decision tree ending blocks are leafs, and the blocks in between are nodes. For a classification task, the important features are normally located at the top few layers. The more important a feature is, the higher layer it is located. However, for a complex classification task, when many key features are available, it is still difficult to decide which feature should be placed on which layer. Although many optimization approaches are proposed to grow an efficient and accuracy decision tree, for complex classification applications, the classification accuracy of a decision tree is relatively low.

Besides, there is no theoretical explanations exist for decision tree-based classification, therefore, the quality of decision tree based classifier heavily relays on the experience of the programmer. In theory, the decision tree can be composed of many layers with hundreds of nodes. However, it is rarely used in real applications. In fact, decision tree is ordinarily used for simple classification tasks or as a weak classifier of boosting algorithm, which will be introduced in the next

section.

2.3.3 Boosting

Boosting is another commonly used supervised machine learning approach in the field of artificial intelligence, statistics, robotics, etc. The principle idea of boosting is to minimize the training error by combining a series of weak classifiers (e.g., decision tree) to a stronger classifier [33]. The most widely used boosting approaches are AdaBoost and Gradient Boosting. In object detection applications, AdaBoost is ordinarily employed. Therefore, more details of AdaBoost is explained in the following paragraphs.

AdaBoost

AdaBoost, which is the abbreviation of adaptive boosting, is a typical boosting method used in object detection algorithms such as [34] and [35]. It combines a series of weighted weak classifiers to the final stronger classifier. As a matter of fact, the training process of AdaBoost is the reweighing process of samples. This reweighing process can be explained as follows. Before data training, all the samples are equally weighted. After each weak classifier is trained, the weights of samples that are correctly classified by this weak classifier will decrease and the weights of the samples that are misclassified will increase [17]. This reweighing procedure ensures that more efforts are drawn to the samples, which are difficult to classify. As long as each weak classifier performs better than chance ($> \frac{1}{2}$), a boosted accuracy of the final strong classifier is guaranteed.

If we denote $\mathbf{x} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^n\}$ as a set of training samples and $\mathbf{y} = \{y_1, y_2, ..., yn\}$ as the labels of the samples respectively. And $W_k(i)$ represents the weight of the i-th sample after the training of k weak classifiers. The computation process of AdaBoost can be described as the Algorithm 1 shown.

In line 7, $h_k(x^i)$ is the label of sample x^i given by weak classifier C_k. In line 8 and 10, Z_k is a constant, which is computed to make sure $W_k(i)$ can represent a probability distribution. The final classification function is obtained by:

$$g(\mathbf{x}) = \sum_{k=1}^{k_{max}} \alpha_k h_k(\mathbf{x}), \tag{2.6}$$

which is the weighted sum of the outputs of all weak classifiers.

Algorithm 1 AdaBoost

initialize k_{max}, $W_1(i) = \frac{1}{n}$, $i = 1, ..., n$
$k \leftarrow 1$
while $k \leq k_{max}$ **do**
    train weak classifier C_k using samples with weights $W_k(i)$
    $E_k \leftarrow$ training error of C_k weighted by $W_k(i)$
    $\alpha_k \leftarrow \frac{1}{2} \ln[(1 - E_k)/E_k]$
    if $h_k(\mathbf{x}^i) = y_i$ (correctly classified) **then**
        $W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} \cdot e^{-\alpha_k}$
    else
        $W_{k+1}(i) \leftarrow \frac{W_k(i)}{Z_k} \cdot e^{\alpha_k}$
    end if
    $k \leftarrow k + 1$
end while
return all C_k and α_k

2.3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is a category of Neural Networks (NN), which has proven very effective in the tasks of object detection, classification, and Natural Language Processing (NLP) [36]. CNN is becoming increasingly prevalent in recent two years. CNN is similar to the ordinary NN with the explicit assumption that the inputs of the system are 2D image data (or any other 2D input such as speech signals) [37].

A CNN is usually comprised of a few convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural networks [38]. The conceptual illustration of a convolutional neural network is shown in Figure 2.11.

In Figure 2.11, five filters are trained and employed at the C1 layer. Each feature map obtained after convolution operation by C1 layer is resampled or processed by other operations at next layer. Layer by layer, raw pixel values of the input image are transformed to the final class scores. Due to five processing lines of Figure 2.11 are independent of each other, five different critical features are extracted by each processing line. (In this work, only one CNN-based algorithm [8] has been studied, adapted, and tested. The theory (and more details) behind CNNs is not introduced since it is out of the topic of this thesis. For more information about

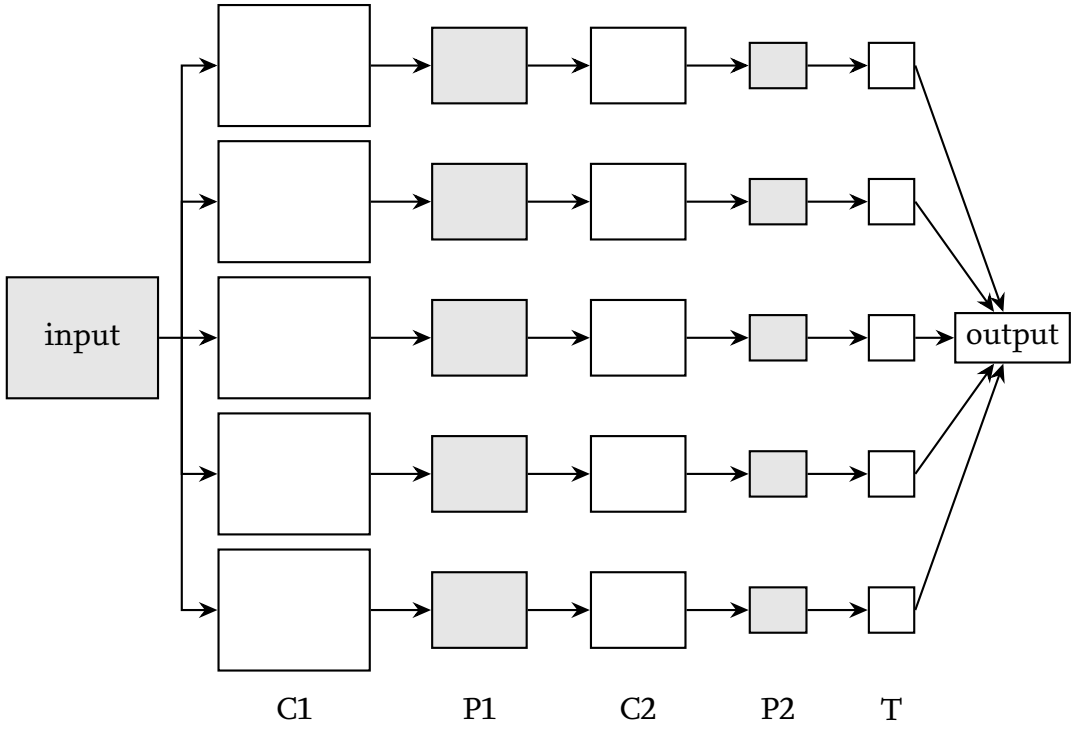


Figure 2.11: The conceptual illustration of a convolutional neural networks

convolutional neural networks and the tutorials, please refer to [39], [40], and [41].)

2.4 Related Work

After introducing the typical features and classification methods used in various object detection applications, in this section, the related works of object detection algorithms and systems are introduced.

Dalal et al. [14] proposed utilizing HoG feature as the descriptor in pedestrian detection. In their work, combining HoG feature and linear-SVM based classifier achieves good performance on pedestrian dataset INRIA [22]. However, the detection speed of HoG plus linear-SVM approach is extremely slow, and the performance of HoG plus linear-SVM decreases on complex datasets (e.g., Caltech-USA dataset [42]). In spite of these drawbacks, comparing to other features employed in pedestrian detection, HoG feature still proved to be one of the best features that can be used to represent an object. Therefore, HoG-like feature which computes HoG with different numbers of bins, different sizes of cells, are proposed, tested and employed. Besides, HoG-like algorithm has also been adapted to a variety of other object detection tasks, such as traffic sign detection [43] [44] and cyclist detection [45]. Today, HoG + linear-SVM is the standard object detection approach

that utilized in OpenCV [46], RoS detection library [47], etc.

To achieve fast object detection, Viola and Jones proposed a framework of rapid object detection in [24]. Thereafter, a robust real-time face detection framework is implemented by Viola and Jones [25]. In their works, a new feature-representing image "integral image" is introduced. "Integral image" is calculated by, for each pixel, computing the sum of pixels that above and on the left of the current pixel. With "integral image", the features are only required to compute once for the whole image. Besides, a "cascade" of weak classifiers are employed as the final classifier in their work. With the rapid object detection framework, 15 fps (frame per second) face detection is achieved for 384×288 images.

Inspired by the detection framework of Viola and Jones, Dollár et al. introduced the integral channel features (ICF) algorithm [19] for pedestrian detection, which pushes both computational efficiency and accuracy to another level. ICF can be considered as a variant of HOG feature. Instead of building the histogram to describe the distribution of gradient orientations, channel images are employed to represent a certain number of orientation bins [19]. It makes the features much easier to be selected by the classifier and easier to be visualized as well. Furthermore, LUV channels from CIE LUV color space [20] are also employed as the feature channels in ICF, which added extra robustness besides the HoG-like feature.

Based on the channel features introduced by ICF, many variants of ICF algorithm (e.g., FPDW [48], ACF [34]) were proposed, and improvements are achieved by these variants. FPDW [48] and VeryFast [35] are two representative works that focus on improving the detection speed of the system. In FPDW [48], a mathematical theory of feature approximation is proposed. With this approximation, feature extraction speed can be dramatically boosted. On the other hand, VeryFast [35] achieves 100 fps pedestrian detection on a modern workstation by using stereo images to locate the range of interest (RoI) area and optimizing the code with CUDA C. With the success of FPDW method, ACF [34] applies the feature approximation method with a better feature representation. Instead of using the integral channel image, the original feature channels are employed in ACF. The results show that better detection performance can be achieved by replacing the integral channel images to the original feature channel images. Moreover, the feature extraction speed of ACF is $7\times$ faster, compared to ICF.

Zhang et al. [6] utilizes different kinds of complex filters to extract features from channel images and achieves good performance on Caltech-USA dataset [42]. The filters are pre-built based on the distinct shapes of human's body parts, such as shoulder and head, which allow the extracted features give the best discriminabil-

ity in classification. However, using these complex filters decrease the detection speed significantly.

In recent two years, great success is achieved by deep learning in some object detection applications. Yang et al. [49] employs convolutional neural network (CNN) to extract features from images, and the ICF's boosting tree method is used as the classifier. The experimental results show that, with their dataset, the log-average miss rate of the CNN-based classification algorithm is around 19%, compared to the ACF is 29%. Meanwhile, the processing speed of ACF on CPU is 0.6 second per frame, and CNN-based algorithm requires 26 seconds per frame on CPU+GPU platform. Tian et al. [50] proposed an algorithm that utilizing joint semantic information, such as human with a backpack, to detect humans. In their work, the single binary classification task is replaced by multi-level classification tasks. For each level classification, one semantic object is detected. They conclude that their methodology improves the detection accuracy and reduces the false positive rate, with the price of significantly increased algorithm complexity.

There are many more object detection applications that utilizing convolutional neural network or its variants to train the classifier, such as [51]. The experimental results show that most of these algorithms can outcome better detection results than the conventional approaches. However, as mentioned above, the detection speed of these deep learning based algorithms are very slow on a modern PC. With the highly optimized convolutional neural network and powerful GPUs, $\approx$10 fps pedestrian detection, $\approx$15 fps face detection can also be reached in [52] and [53], respectively.

In general, all the research works that mentioned above are focused on improving the detection performance on a PC platform. Meanwhile, there are many researchers devoted their efforts to optimizing various detection applications on PC+GPU, embedded system, or FPGA platform. A face detection system [54] is designed and implemented on a FPGA based on the algorithm proposed by Viola and Jones [25]. With the low-end Virtex-II FPGA, similar detection performance and speed are achieved, compared to the algorithm running on a high-end computer. However, the power consumption of the system on the FPGA is 20 times less than on a PC. Suleiman et al. [55] designed an ASIC architecture to realize the HoG+linear-SVM based pedestrian detection. The results show that 60 fps pedestrian detection can be reached with low-power consumption. Benenson in [35] optimized the object detection application ICF on a CPU+GPU platform. With the deep optimized CUDA C code, 100 fps object detection can be achieved for 640×480 images.

Kyrkou et al. [56] implement a general object detection framework by employing "integral image" on a FPGA. "Integral image" based feature calculation process is faster than conventional computation process on a PC platform. As a result, its implementation on the FPGA platform is even faster. According to their results, 2 to 4 times speedup is achieved. Similarly, an "integral image" based traffic sign detection system is implemented in [57]. In their work, the original image is downsampled 4 times. Therefore, 5 times of detection under different image scales are performed for each input image. Its final implementation on a ZC706 evaluation board reaches 126 fps traffic sign detection.

Besides, some convolutional neural networks based object detection applications are implemented on hardware platforms as well. Janus implements a car detection system by employing customized CNN on a FPGA in [58]. Because object segmentation approach is employed, 30 fps car detection for the 1280×720 image is achieved. For some more complex detection tasks, such as YOLO [59] which requires 13 seconds to process per image, the deep learning based object classifier can achieve 0.65 second per image processing on a ZC706 FPGA. Although it is far from real-time, compared to the CPU version, the detection speed is 25 times faster.

2.5 Summary

In this chapter, the typical structure of object detection algorithm/system is introduced. Then, some commonly used features and classification methods are introduced. After that, the state-of-the-art algorithms and systems of object detection are described.

Chapter 3

Implementation and Comparison of Object Detection Algorithms

Ordinarily, one object detection algorithm is targeting one category of objects, such as faces, humans, and cars. Due to the variety of humans, pedestrian is employed as the example detection object in this work. Therefore, in this chapter, the most critical and state-of-the-art pedestrian detection algorithms are selected, implemented, and compared. After that, the detection results of implemented algorithms are analyzed.

3.1 Selection of Object Detection Algorithms

To realize object detection, the first task is choosing the targeting category of objects. In this section, the application target and its representative algorithms are selected.

3.1.1 Application Selection

In the last two decades, object detection applications that target various instances of objects are proposed, such as traffic sign detection, front-car detection, and pedestrian detection, as illustrated in Figure 1.1. Among these applications, pedestrian detection is one of the most difficult tasks due to the following reasons:

- Firstly, pedestrians are always with different colors of clothes. Colorful clothes make it easier for one person to be recognized by the others. However, it also makes the detection task tougher for a machine.
- Secondly, pedestrians behave with different gestures and poses in captured images.
- Thirdly, due to the changeable temperature on earth, the styles of clothes are also different. For instance, in summer, short sleeves and shorts are worn.

- Moreover, the skin color, the color of hair, and the size of human are also different.

Owing to the reasons listed above, it is challenging to summarize the common features from a variety of pedestrian samples. Therefore, pedestrian detection is selected as the example application in this chapter.

3.1.2 Algorithm Selection

After pedestrian is selected as the detection object, different pedestrian detection algorithms are researched. As mentioned in Section 2.4, the most widely used pedestrian detection algorithm is HoG + linear-SVM [14], which is the baseline for many other detection algorithms. In this thesis, HoG + linear-SVM is also selected and implemented as the baseline detector for performance comparison.

The Integral Channel Features (ICF) algorithm, which is proposed by P. Dollar [19], utilizes the Haar-like feature "integral image" as the feature channel and LUV of CIE LUV color space as extra channels in pedestrian detection. Since most of the conventional pedestrian detection applications are the variants of the ICF algorithm, ICF is selected and implemented in this chapter.

To improve the detection speed, stereo cameras are employed in VeryFast [35]. In this algorithm, the region of interest (RoI) is located by calculating the "UV-disparity" from stereo images. Thereafter, an ICF-like detection process is performed to detect pedestrians in the RoIs of the image. Because of its efficient RoI mechanism, VeryFast algorithm is also implemented in this thesis.

The other works that attempt to speed up the feature extraction process are FPDW [48] and ACF [34]. They introduced a novel approach of feature calculation by approximate computing in the image pyramid. With this strategy, $2 \sim 5$ times speedup of feature extraction can be achieved. Therefore, FPDW and ACF are also realized and compared. Besides, in order to evaluate the effect of different classification methods, "ACFSVM", which consists of the SVM classifier and the feature extraction module of ACF, is programmed and tested.

Due to the popularity of deep learning in object detection, the convolutional neural networks based pedestrian detection algorithm DeepPed [8], which provides one of the best performance of pedestrian detection in Caltech-USA dataset [42], is also selected and implemented in this section.

3.2 Datasets for Pedestrian Detection

Dataset is the necessary component for all object detection applications. There are many different pedestrian datasets provided by researchers and companies. In this work, the most commonly used ones are selected and employed, which are listed as follows.

3.2.1 INRIA Dataset

INRIA dataset [22] was originally provided by Dalal in 2005 for the HoG+SVM work [14]. In this dataset, 1237 positive samples with annotations and 1218 negative pictures are provided for training. 589 annotated positive samples are provided for testing. All positive samples are scaled and trimmed to the size of 128×64, which can be utilized as the size of the sliding window during detection. The dimension of the negative pictures is 640×480. Because the size of each negative sample is 128×64, thousands of negative samples can be generated from each negative picture. Besides, some of the images in this dataset are stereo. Hence, this dataset is used in the VeryFast algorithm [35].

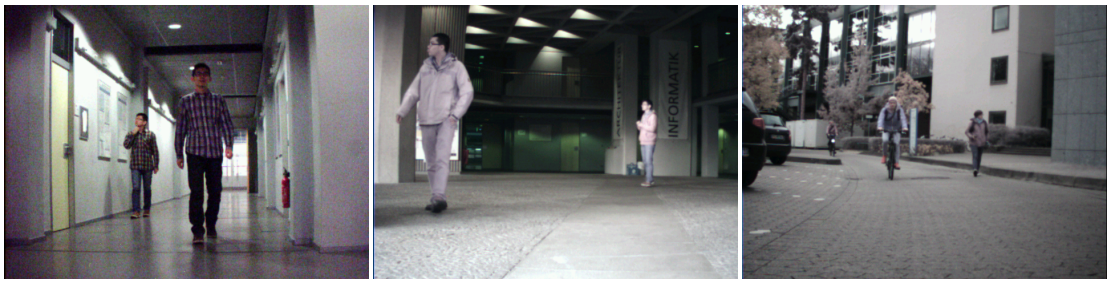
3.2.2 Caltech-USA Dataset

Caltech-USA dataset [42] is one of the biggest pedestrian dataset available online. It contains around 10 hours of 30 fps, 640×480 video that captured by a camera on a driving vehicle. Due to that all images are captured on the car, the pedestrians are smaller, and most of the images have no pedestrians inside. Therefore, this dataset is only utilized to train the generalized object detection algorithm in Section 4.2.

3.2.3 TUBS-C3E Dataset

Except for the public pedestrian datasets, around 150 stereo images and 200 single images are captured by an OV7670 camera. OV7670 is the camera that deployed on the object detection platform of this thesis. (The detailed information about this platform will be introduced in Section 6.1.) Since the same camera is used in the final object detection system, it is meaningful to utilize a few images captured by this camera to train the detector. Figure 3.1 illustrates some examples of the images captured by our camera, which includes both indoor and outdoor scenarios with different lighting conditions.

Following the same annotation criteria, all images of the TUBS-C3E dataset are annotated manually in MATLAB. The 150 pairs of stereo images are utilized in



(a) Indoor with bright light (b) Indoor with natural light (c) Outdoor

Figure 3.1: Illustrations of captured images in TUBS-C3E dataset

the VeryFast algorithm [35] implementation and debugging/verifying the depth calculation module of Section 5.4. Besides, some pictures without pedestrian are also captured, which are employed as the negative pictures during the training phase.

In practical, all samples of the INRIA dataset and TUBS-C3E dataset are used during the implementation of selected pedestrian detection algorithms. The negative samples provided by INRIA dataset and OV7670 captured are combined as the negative dataset for classifier training. All annotated images of two datasets are merged and categorized into training and testing dataset.

3.3 Algorithm Implementation

After the algorithms are selected and the datasets are prepared, in this section, all selected algorithms are implemented on a PC with MATLAB. To obtain better performance and make a comparison, modifications and optimizations of selected algorithms are performed.

3.3.1 Implementation

ACF/ICF/FPDW: Because the ACF algorithm is provided as an open source MATLAB toolbox [60] by P. Dollar, it is directly used in this thesis. By employing some functions of the ACF toolbox, ICF and FPDW, which are the variants of ACF, are realized according to their research papers [19] and [48], respectively.

HoG+linear-SVM: Although HoG is provided as a standard object detection function in OpenCV library [46], to make an equal comparison, a MATLAB version of HoG feature extraction is created and realized. Moreover, to make the HoG feature similar to the gradient orientations feature of ACF, instead of using the original

HoG feature, a HoG-like feature with six bins is implemented in this section. After the six-bin-HoG-like features are extracted from all positive samples, these data are used to train the SVM classifier. During this process, the open source library LIBSVM [61] is employed.

VeryFast: The source code of VeryFast algorithm [35] is provided by Rodrigo Benenson [62]. However, many functions of VeryFast are written in CUDA C [63] and deeply optimized for GPU implementation. Due to the limitation of experimental environments, after analyzing and understanding the algorithm theory and source code, the VeryFast algorithm is rewritten in MATLAB. In the rewrote version of VeryFast algorithm, some functions, such as gradient magnitude computing and histogram of orientations, are realized by using the ACF toolbox [60].

DeepPed To evaluate the performance of convolutional neural networks based DeepPed algorithm [9], all prerequisites that required by DeepPed are installed, which includes the most widely used open source deep learning framework Caffe [64] and a region-based convolutional neural network algorithm R-CNN [65]. Thereafter, configuring the DeepPed with the paper provided settings [9], DeepPed based pedestrian detection is tested and evaluated.

3.3.2 Classifier Training

In the training phase of SVM, the bootstrapping strategy is used. It is known that two stages of classifier training are normally performed in linear-SVM. In this work, a two-stage training of classifier is also performed during the training phase of linear-SVM. In the first stage, the training dataset contains all positive samples and 10000 negative samples that randomly cropped from negative images. When the training is done, a trained SVM classifier is obtained. Then, to further improve the performance of this classifier, the second stage of training is performed by adding "hard negative" samples to the training dataset. Executing detection on negative images with the first stage classifier, many false positive results will be obtained. These false positive detections actually are the "hard negative" samples for the second stage of training. In this work, 5000 "hard negative" samples are added to the second stage training dataset. After these two stages of training, the final SVM classifier can be obtained.

In AdaBoost based algorithms, such as ICF, ACF, and FPDW, similar training strategy is employed. However, due to the depth of each decision tree is two and the total number of trees is uncertainty, multiple stages of bootstrapping are employed during the training phase. Similar to training the SVM classifier, all positive samples and 5000 negative samples are used for the first stage of training. Then, 5000

"hard negative" samples are added to the training dataset in each stage. Worth to note that 30000 features are used in ICF while 5120 features in ACF. Correspondingly, the ratio of positive training samples versus negative samples has to be changed according to the number of the features employed.

3.4 Results Comparison and Analysis

After the selected algorithms are realized or modified, in this section, the performance of these algorithms are given, compared, and analyzed.

3.4.1 Classifier Training Stages

As mentioned in Section 3.3.2, while training the SVM or AdaBoost classifier, multiple-stage of training is executed. For linear-SVM, two stages are normally enough. However, for AdaBoost approach, the number of trees and the number of training stages that required are uncertain. In this section, different settings of tree numbers and stages are evaluated with the ICF algorithm. The performance of the ICF classifiers that trained with different settings are listed in Table 3.1.

Stages	Decision trees	Negatives at each stage	Miss rate
3	[32 256 1024]	[5000 10000 12797]	27.61%
3	[32 512 2048]	[5000 10000 11486]	21.81%
4	[32 128 512 1024]	[5000 10000 15000 17950]	26.02%
4	[32 128 512 2048]	[5000 10000 15000 15988]	18.79%
4	[64 256 1024 2048]	[5000 10000 13556 14395]	19.41%
4	[2048 2048 2048 2048]	[5000 8710 10089 10419]	23.71%
5	[32 128 512 1024 2048]	[5000 10000 13556 14036]	18.81%
5	[64 256 512 1024 2048]	[5000 10000 13756 14917]	19.13%

Table 3.1: The performance of ICF classifiers trained with different number of training stages, trees used at each stage, and total number of trees

The third column of Table 3.1 listed the negative samples used at each training stage. Except for the first stage, maximum 5000 "hard negative" samples are detected by the n-st stage classifier and added to the next round of training. From the number of negatives added at each stage, we can defer that when the number of "hard negatives" added to current stage of training is limited, the potential for further performance improvement is also limited.

In Table 3.1, the performance of classifiers with the total number of 1024 and 2048 trees are listed. With 1024 trees, the log-average miss rate of the ICF clas-

sifiers is 26% $\sim$ 28%, while with 2048 trees, 5% $\sim$ 8% performance enhancement is achieved. Moreover, the detection results of ICF classifier trained by 3-stage, 4-stage, 5-stage are also given in Table 3.1. By comparing the results of 3-stage, 4-stage, 5-stage trained classifiers, we can notice that the performance of 4-stage trained classifier is around 3% better. However, similar detection results are obtained from 4-stage and 5-stage trained classifiers. Besides, the six line of Table 3.1 shows the performance of the classifier trained by four times of 2048 trees. After the first stage of training, 4110 "hard negatives" are detected and added to the second stage of training. However, the result of final classifier only reaches 23.71%, which is much worse than other 4-stage trained classifiers. One possible reason is that, due to too many trees utilized in the 1st and the 2nd stage of training, too less negative samples are added to the 3rd and the 4th stage of training.

3.4.2 Features Versus Classification Methods

Among the pedestrian detection algorithms that selected in Section 3.1.2, two classification methods are employed, namely, support vector machine and AdaBoost trees. Therefore, the impacts of two different classification methods are compared. (Because convolution neural network does not require feature extraction, it is too different comparing to the other algorithms. Hence, it is not compared in this section.)

To accomplish this comparison, HoG+linear-SVM and HoG+AdaBoost are implemented in MATLAB. Although HoG+SVM based pedestrian detection is provided in OpenCV library, for the ease of comparing with other algorithms later, the ACF detection toolbox provided by P. Dollar [34] is employed. In this toolbox, AdaBoost decision tree training section is available. Moreover, the functions to calculate gradient magnitude and orientations are also provided. Therefore, based on these functions, HoG feature is extracted. After that, a HoG+AdaBoost algorithm is realized and evaluated. In HoG+AdaBoost algorithm, the AdaBoost is configured with 2048 2-depth decision trees, 2436 positive samples, and 15000 negative samples. On the other hand, for the SVM-based classifier, the tool LIBSVM [61] is employed during the training phase.

Besides, since feature extraction part of ACF algorithm is available in Dollar's toolbox, ACF+AdaBoost and ACF+SVM is also implemented. After training and testing with different configurations for multiple times, the best performance achieved by these four algorithms are shown in Figure 3.2.

Figure 3.2 is a typically used method to evaluate the performance of different algorithms, which is called Receiver Operating Characteristic (ROC) curve. The x-axis

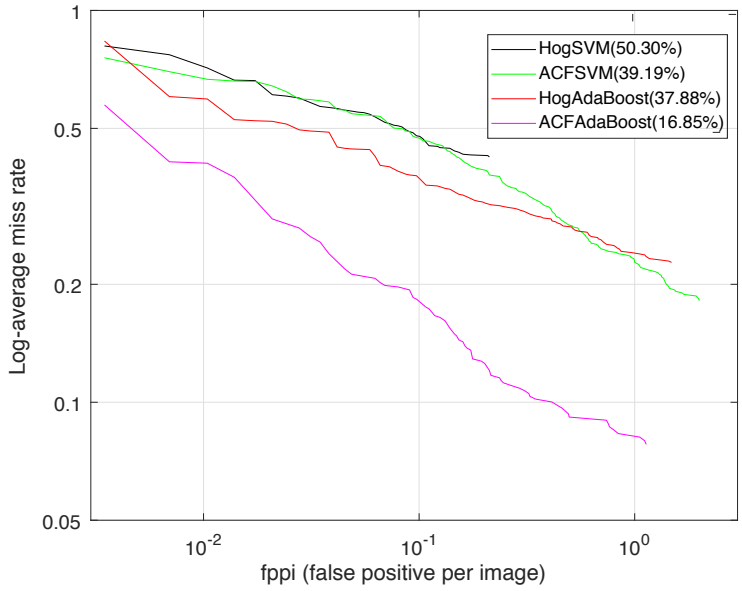


Figure 3.2: The performance of different classification methods with ACF and HoG descriptors

represents the false positive in each image, which is equal to divide the number of false positives by the number of images in testing dataset. The miss rate of the classifier is shown in the y-axis. For the ease of comparison and display, the log-average miss rate is calculated and utilized in the RoC curve. With the understanding of x, y-axis meanings, it is clear that better performance is achieved when the RoC curve is closer to the lower left corner of the coordinate system.

Comparing the ROC curves of Figure 3.2, we can notice that by replacing the SVM classification method with AdaBoost, 12%∼22% performance improvement can be achieved. One possible reason is that 2-depth decision tree in AdaBoost provides two layers of weak classifiers, while the linear-SVM is similar to a group/cascade of one layer of weak classifiers. Meanwhile, comparing the detection miss rate of HoG+AdaBoost and ACF+AdaBoost, it is evident that 20% performance enhancement is also achieved by using better feature representatives. Although only HoG and ACF are compared in this section, we can infer that comparing to HoG+linear-SVM, better performance can also be obtained by ICF, FPDW, etc. ACF-like algorithms.

3.4.3 Evaluation of Speed Improvements

Among the selected algorithms, FPDW [48] and VeryFast [35] focus on improving the detection speed of ICF-like algorithms. In FPDW, the detection speed is im-

proved by simplifying the computation complexity of feature extraction process, while VeryFast achieves speed enhancement via reducing the ROIs by utilizing the stixel information from the stereo images. Since both approaches can also be employed in the hardware design of the object detection system, therefore, the speed improvements of FPDW and VeryFast are evaluated in this section. Besides, since the same feature calculation strategy as FPDW is employed in ACF algorithm, the detection speed of ACF algorithm is also evaluated.

In the original ACF algorithm, the LUV channel features are preprocessed by an 11×11 triangle filter before HoG-like feature computation. However, in this experiment, the size of triangle filter is set to 3×3. In order to make a fair comparison, the same triangle filter is added to FPDW, ICF, and VeryFast algorithms. Meanwhile, a version of each algorithm without triangle filtering is also implemented and evaluated in this part. The detection speed of all evaluated algorithms is listed in Table 3.2.

Algorithm	Running speed (frame per second)	
	Without triangle filtering	With triangle filtering
ICF	4.12	3.90
FPDW	10.79	9.37
VeryFast	11.55	10.06
ACF	13.79	11.97

Table 3.2: The detection speed of different pedestrian detection algorithms

All algorithms of Table 3.2 are evaluated on a computer with an Intel i7-3770 CPU and a NVIDIA Geforce GTX 750 Ti GPU. A dataset consists of INRIA and TUBS-C3E is utilized as the dataset for all evaluated algorithms. From Table 3.2, we can notice that the detection speed is dramatically improved by FPDW, VeryFast, and ACF. Although same feature approximate computing strategy is employed in FPDW and ACF, the detection speed of ACF algorithm is 30% faster than FPDW, which is caused by the following reasons:

- Firstly, better feature representative is employed in ACF algorithm. In ICF and FPDW, the Haar-like "integral image" of features is used. However, the original features are directly used in ACF.
- Secondly, 30000 random rectangle features are selected to train the detection classifier in FPDW. Meanwhile, 5120 features of fixed size are used in the ACF.

On the other hand, VeryFast algorithm also achieves dramatical improvement on

detection speed by utilizing stereo images. Nevertheless, to obtain the stereo information provided by stereo images, ground plane estimation and stixel calculation [62] processes, which also require an enormous amount of computation resources, are added to the VeryFast algorithm. Therefore, the final detection speed of VeryFast is slower than ACF. It is also worth pointing out that all stereo images used in the VeryFast algorithm are after rectification. In other words, the image undistortion and image rectification parts are not included in the VeryFast algorithm yet.

With respect to the impact of filtering, since only a 3×3 triangle filter is added to each algorithm, slightly slower of pedestrian detection is caused by this triangle filtering process.

3.4.4 Performance Comparison and Analysis

After multiple times of training and testing with different configurations for each algorithm, all selected algorithms are eventually implemented, optimized, and evaluated. The ROC curve of each pedestrian detection algorithm is drawn and shown in Figure 3.3.

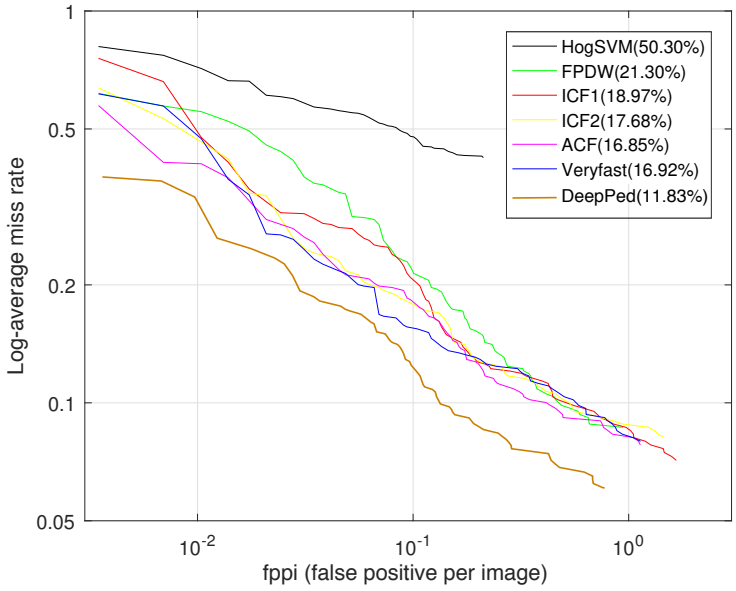


Figure 3.3: Performance comparison of selected detection algorithms on INRIA + TUBS-C3E dataset

From Figure 3.3, we can conclude that the best detection performance is achieved by the convolutional neural networks based DeepPed approach. Meanwhile, similar detection accuracy is obtained by the ACF and the VeryFast algorithms. In

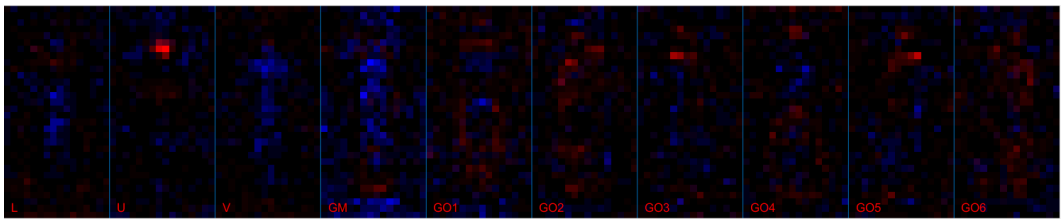
order to analyze the differences among the selected algorithms, the visualization of all the data that used in each algorithm's classifier is drawn in Figure 3.4.

Due to that ten types of features are extracted and utilized in the ACF, ICF, and VeryFast algorithms, all feature data employed in the trained classifier are visualized in ten channels, which is shown in Figure 3.4. In this figure, the features that chose and used in the classification process marked in red and blue colors. The features with red marker indicate positive contributions to the final confidence value and the features with blue marker indicate negative contributions to the final confidence value. Both red and blue features are important to comprise a detector. The brighter a feature is, the more important this feature is for classification.

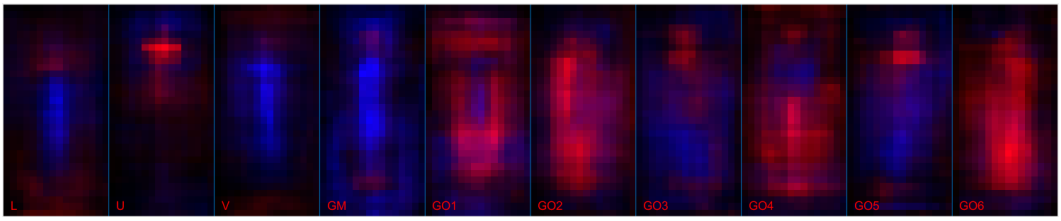
By comparing the L, U, and V channel images of Figure 2.3 and the L, U, and V channel visualizations of Figure 3.4(a), it is evident that the key features in L, U, and V channel images match the bright red area of Figure 3.4(a). In other words, the ACF algorithm successfully locates the key features with a promising discriminability. Moreover, through comparing the feature data of gradient magnitude channel and histogram of gradient orientation channels, which are shown in Figure 2.6, and the GM (gradient magnitude), GO1 (histogram of gradient orientations), ..., GO6 subfigures of each visualization in Figure 3.4, we can discover that key features of gradient magnitude and hog-like channels are also successfully extracted.

Figure 3.4(b) is the visualization of the original ICF classifier, whose features are 30000 randomly selected any-sized rectangles of the "integral image." It is obvious that Figure 3.4(b) is much more colorful than Figure 3.4(a). However, the performance of the ICF algorithm is worse than the ACF algorithm, as shown in Figure 3.3. Therefore, we can infer that the key features used in the ICF classifier actually are worse than the ones used in the ACF classifier. From the ACF algorithm [34], we know that the size of all features is the same and equal to 4×4. Considering the difference of feature image and "integral image," another version of ICF classifier that consists of 30000 features where the size of each feature is smaller than 16×16 is implemented and denoted as ICF2. The visualization of the ICF2 classifier is also generated and shown in Figure 3.4(c). As can be seen from 3.4(c), the feature representations used in ICF2 classifier is much sharper and clear than the original ICF classifier. Therefore, theoretically, better performance can be obtained by the ICF2 classifier.

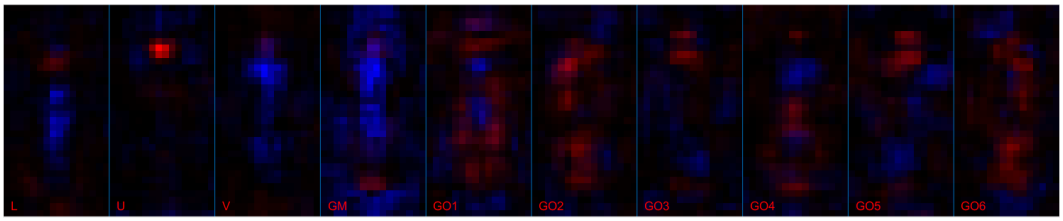
To verify this conjecture, the ICF classifier with different sized features are trained and tested. The experimental results are listed in Table 3.3. From this table, we can



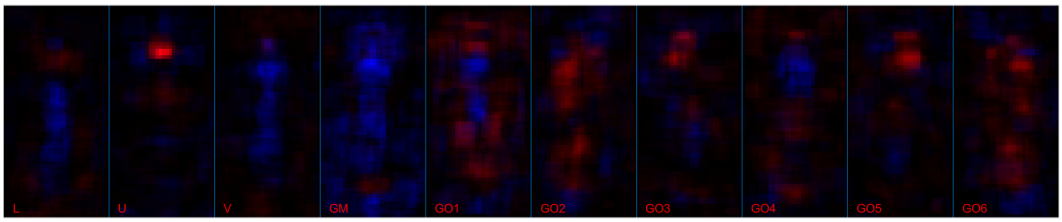
(a) Visualization of features used by the ACF classifier



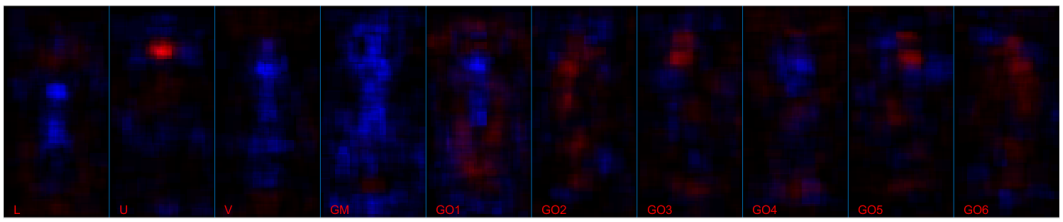
(b) Visualization of features used by the ICF1 classifier (ICF with 30000 different-sized rectangles)



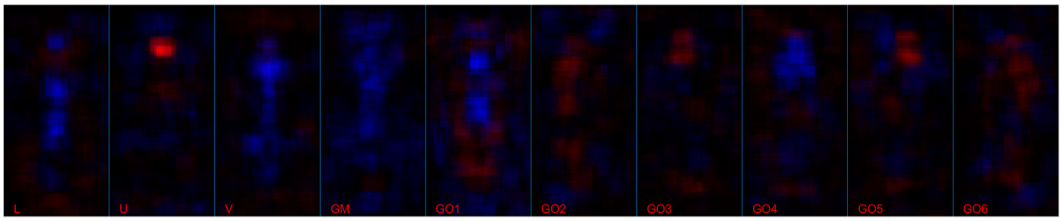
(c) Visualization of features used by the ICF2 classifier (ICF with 30000 rectangles whose size is smaller than 16×16)



(d) Visualization of features used by the VeryFast sub-classifier '128×64'



(e) Visualization of features used by the VeryFast sub-classifier '256×128'



(f) Visualization of features used by the VeryFast sub-classifier '512×256'

Figure 3.4: Visualization of the classifiers used in different algorithms

attain that when the maximum size of the feature decreases, the ICF classifier with better performance can be obtained. Which is to say, features from large rectangle do not have much impact during the classification process. The possible reason of this could be large rectangle features make characteristics in channel images more difficult to be represented. On the other hand, when the maximum size of the rectangle is too small, too many repetitive features exist in the 30000-feature pool. Therefore, 16×16 is the proper configuration that employed in the final implementation of the ICF algorithm.

Classifier	Maximum size of each feature	Log-average miss rate
ICF	Any Size	18.97%
ICF	16×16	17.68%
ICF	8×8	17.62%
ACF	4×4	16.85%

Table 3.3: Performance comparison of the ICF classifiers that trained with different configurations (configurable parameter: the maximum size of features employed)

As mentioned in Section 2.4, the training and detection phases are different from the other ICF-like algorithms. In the VeryFast algorithm, the training images are rescaled to different resolutions, and the detection is executed only on the original image. Due to different sized pedestrians in the captured image, many sub-classifiers are eventually obtained after the training phase; each sub-classifier is responsible for detecting pedestrians with a specific size. Figure 3.4(d), 3.4(e), and 3.4(f) are the visualizations of three exemplified sub-classifiers of the VeryFast classifier.

As can be seen from three visualizations, almost identical characteristics are selected and employed by each sub-classifier. However, in practical, the size of three illustrated classifiers are quite different, which cannot be seen from Figure 3.4 owing to the limitation of the paper size. Figure 3.4(d) shown sub-classifier '128×64' is generated to detect the pedestrians around the size of 128×64. Similarly, Figure 3.4(e) and 3.4(f) shown sub-classifiers are employed to the detect the pedestrian whose size is similar to 256×128 and 512×256 in the image, respectively. Because many sub-classifiers are used during the classification phase, different thresholds are configured for each sub-classifier. Hence, promising performance is achieved by the VeryFast classifier, as shown in Figure 3.3.

3.5 Summary

In this chapter, to evaluate the performance of different object detection applications, pedestrians are selected as the targeting object. After that, critical and state-of-art pedestrian detection algorithms are chosen and implemented on a PC platform. In the end, the performances of different detection algorithms are compared and analyzed.

Chapter 4

Framework of the Generalized Object Detection System

The main objective of this thesis is to design and implement different object detection systems on a low-power SoC-FPGA platform. Nevertheless, most of the object detection algorithms can only detect one category of objects, e.g., cars, faces, pedestrians. In order to achieve a generalized object detection system on SoC-FPGA, in this chapter, the framework of the generalized object detection algorithm is proposed. With this framework, different instances of object detection applications, such as pedestrian detection and traffic sign detection, are implemented and verified.

4.1 The Generalized Object Detection Framework

As shown in Figure 3.3, the CNN-based DeepPed algorithm [8] achieves the best performance among all implemented object detection algorithms of Chapter 3. However, the CNN-based object detection algorithm consists of multiple layers of convolution operations, where each convolution operation employs a different sized model, e.g., 3×3, 7×7, 11×11. Moreover, when the classifier re-trained each time, the size of the convolution model on each layer also varies. The diversity of employed convolution models and the variability of convolution operations after re-training on each layer increase the difficulty of designing a universal and efficient convolution module on a SoC-FPGA platform.

Besides, CNN-based object detection algorithm achieved promising detection accuracy is usually on the basis of more convolution operations and larger convolution models, which require a large amount of resources to implement on FPGA. Considering the targeting platform of this work is Zedboard [66], whose programming logic resources are finite, CNN-based object detection approach is not employed to compose the generalized object detection framework. After some experimental tests and comparison, an optimized ACF algorithm is ultimately employed

as the main component of the generalized object detection framework. The fast feature calculation strategy, which provides extra speed enhancement during the feature extraction process and requires evidently less computing resources, compared to the VeryFast algorithm [62] and DeepPed algorithm [8], is utilized in this framework. Furthermore, compared to other pedestrian detection algorithms implemented in Section 3.4.4, the ACF algorithm achieves promising results on the pedestrian detection task.

In the following sections, the framework of the generalized object detection system is introduced. In addition, to design the feature extraction module by using fast feature pyramid approach, the theory behind fast feature calculation and relating configurations are also explained.

4.1.1 The Framework Overview

The overall structure of the generalized object detection system is shown in Figure 4.1. Because this system targets to detect different categories of objects, the data training and testing phases on a PC with MATLAB are required. Moreover, to generate the best classifier for each application (e.g., face detection application, traffic sign detection application), the training and testing phases are executed multiple times with different configurations until a classifier with promising performance is obtained.

As can be seen in Figure 4.1, the trained classifier, which achieves the best performance in training phase, is exported to the FPGA platform. It is worth noting that the feature extraction process for training images and testing images are different. For the training images, all positive samples are resized to the size of the sliding window. For instance, 128×64 is used for the pedestrian detection application. Hence, rescaling or image pyramid processing is not required. However, for the testing phase, each image is rescaled multiple times to detect different sized objects. Therefore, a fast feature calculation process is employed and denoted as the 'Feature Extraction' module in Figure 4.1. For the feature calculation process under each image scale, the module 'Feature Calc' is used and denoted.

After the classifier data is exported, the object detection is executed on the FPGA platform. In order to achieve real-time object detection, stereo cameras are employed as the data input of the system. The feature data are extracted from images of the left camera by the fast feature pyramid method, which is the same as the feature extraction process of testing phase on the PC. When the feature data are ready, classification is performed by employing the trained classifier with a sliding window. The output from classification is the location information of the ob-

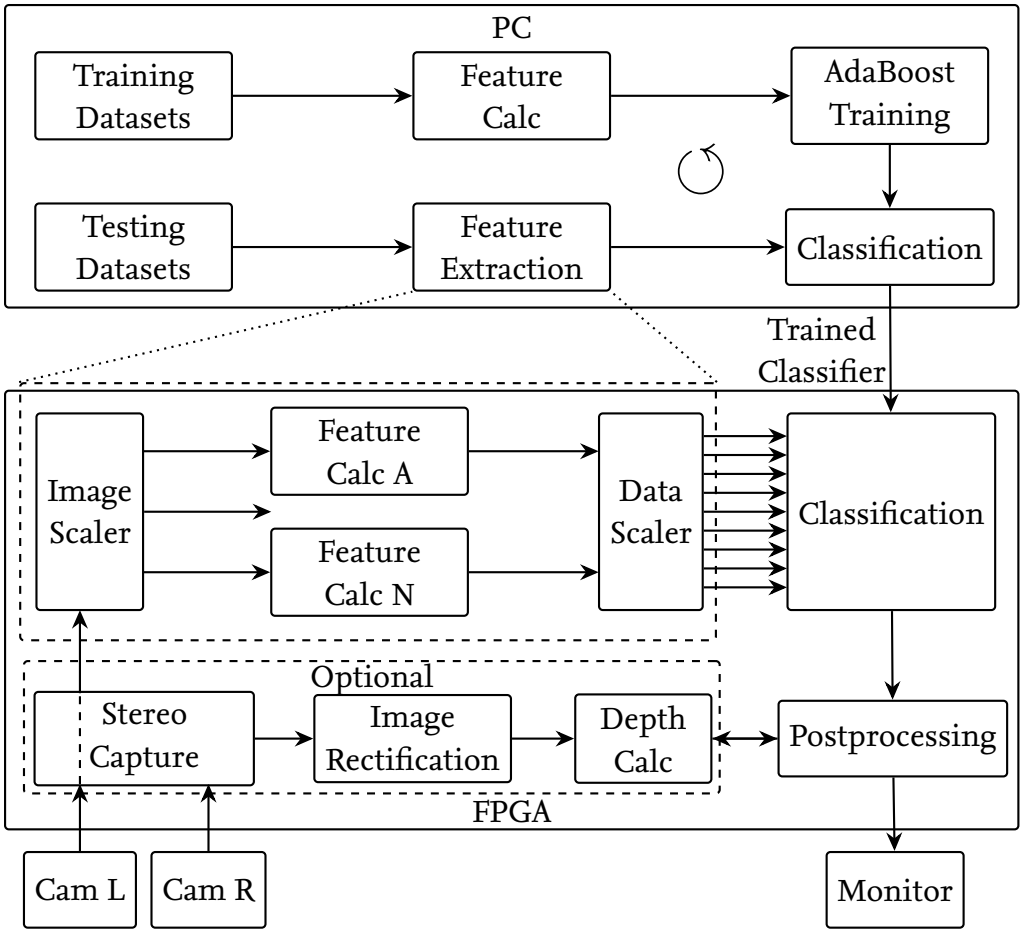


Figure 4.1: The framework of the generalized object detection system

jects detected by the classifier. However, due to duplicate objects are detected by nearby sliding windows or under nearby image scales, the post-processing module is added to remove the overlapped detection results. Finally, The image with marked objects is shown on a monitor. Optionally, the depth of each detected object can be calculated with stereo image rectification and depth calculation modules.

4.1.2 Feature Extraction by Fast Feature Pyramid

In object detection applications, to detect different sized objects in the image, the input image is rescaled multiple times, which is called image pyramid. The conventional feature extraction process of the image pyramid is shown in Figure 4.2. The input image is rescaled to different sizes by image scaling. Then, the feature data are extracted for each scaled image. Because complex computing process is performed inside each feature calculation block, a huge amount of comput-

ing resources are required to accomplish the whole feature extraction process for multiple image scales.

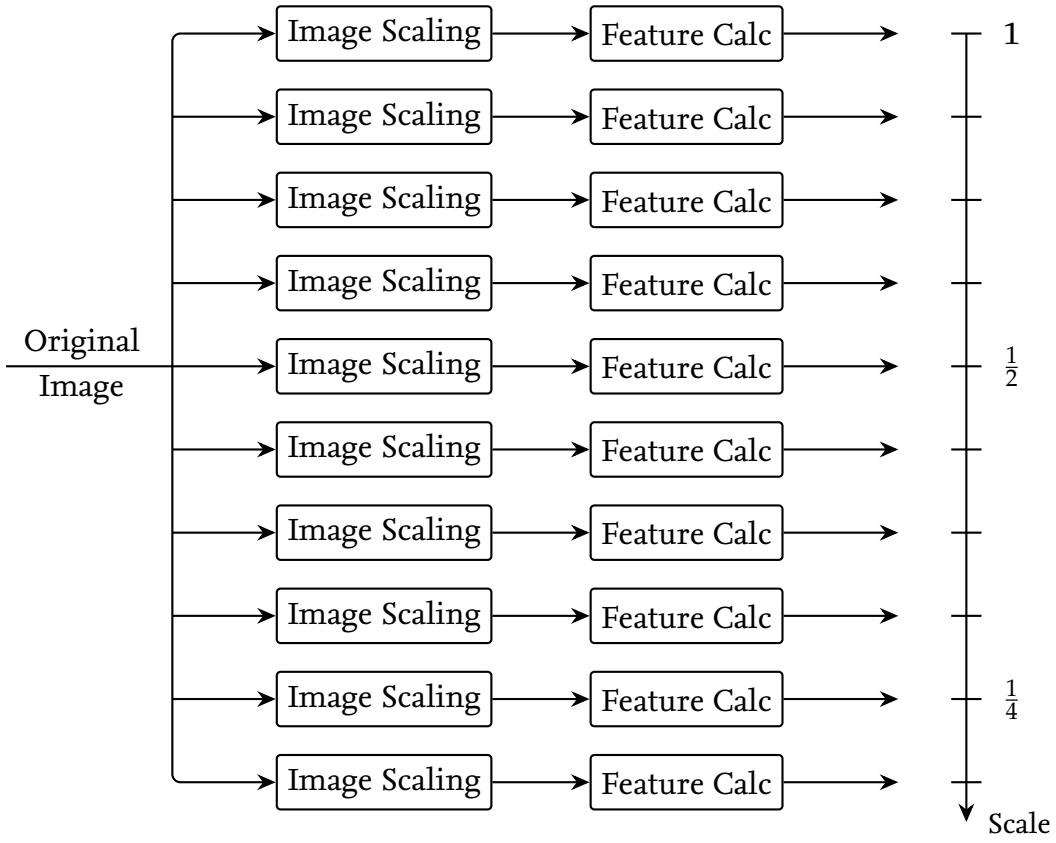


Figure 4.2: The traditional method of feature extraction from image pyramid

Ruderman and Bialek [67] discover that the statistic of resized natural images follows a power law, which can be expressed to

$$E[\Phi(I_{s_1})]/E[\Phi(I_{s_2})] = (s_1/s_2)^{-\lambda_\Phi}, \tag{4.1}$$

where $\Phi(I)$ is an arbitrary image and $E[\cdot]$ is the expectation over a patch of natural images. s_1 and s_2 are the scales of this patch of a natural image. From Equation (4.1), we can derive that the ratio of $E[\Phi(I_{s_1})]$ and $E[\Phi(I_{s_2})]$ depends on the ratio of s_1/s_2 and a constant parameter λ_Φ. Dollár in [34] figure out that similar expression is also valid for the channel images. If we denote any channel image as $f_\Omega(I)$, the Equation (4.1) can be rewritten as

$$f_\Omega(I_{s_1})/f_\Omega(I_{s_2}) \approx (s_1/s_2)^{-\lambda_\Omega}, \tag{4.2}$$

where I_{s_1} and I_{s_2} are the image I at scale s_1 and s_2, respectively. For the ease of description, assuming the scale s_2 is equal to 1, then, I_{s_2} is the original image and

$f_\Omega(I_{s_2})$ is a channel image (e.g., L channel image, gradient-magnitude channel image) calculated from the original image. Therefore, the channel image $f_\Omega(I_{s_1})$ can be calculated by

$$f_\Omega(I_{s_1}) \approx s_1^{-\lambda_\Omega} \cdot f_\Omega(I_{s_2}). \tag{4.3}$$

In other words, when the channel features of the original image $f_\Omega(I_{s_2})$ is obtained, the channel features at scale s_1 can be calculated by $f_\Omega(I_{s_2})$ multiplying $s_1^{-\lambda_\Omega}$, where λ_Ω is a constant parameter. In fact, this approximation relation is also valid for any s_1 and s_2 scales.

To verify the accuracy of the approximation expressed in Equation (4.2), comparisons are made between the results of normal computing and approximate computing by Equation (4.2). The results reveal that the approximate computing method by Equation (4.2) is more accurate when the ratio of s_1 / s_2 is in the range of [0.5, 2]. However, in practical detection applications, the image scales that smaller than 0.5 of the original image is doubtless required. Therefore, certain times of real computing and approximate computing based on each real computing are performed, which is illustrated in Figure 4.3.

In different object detection applications, due to the size-difference of detection objects, the size of the sliding window is different. Hence, the amount of required real computing units also varies. For instance, the size of sliding window employed in pedestrian detection application is 128×64 and the resolution of the input image is 640×480. In order to detect all of the different sized pedestrians from the input image, the smallest scale equals $\frac{128}{480} (\approx 0.25)$. Since the approximate computing method is more reliable when the scales are in the range of [0.5, 2]. Therefore, real computing is employed to realize feature calculation at scale 0.25. In reality, three real feature calculation blocks are used for the pedestrian detection application. After the real computing, the feature approximation is employed to calculate the surrounding scales of each real computing scale, as shown in Figure 4.3. Because only bilinear interpolation is executed in each approximate computing unit, compared with the conventional method that shown in Figure 4.2, much less computing resources are required.

Until now, the parameter $-\lambda_\Omega$ of Equation (4.2) is not mentioned. Therefore, the next question is how to calculate this constant parameter. The theory behind this parameter calculation is complicated and will not be introduced in this section since it is out of the topic of this thesis. (The theory of calculating the constant parameter $-\lambda_\Omega$ can be found in [34].) However, the calculation process can be summarized into three steps:

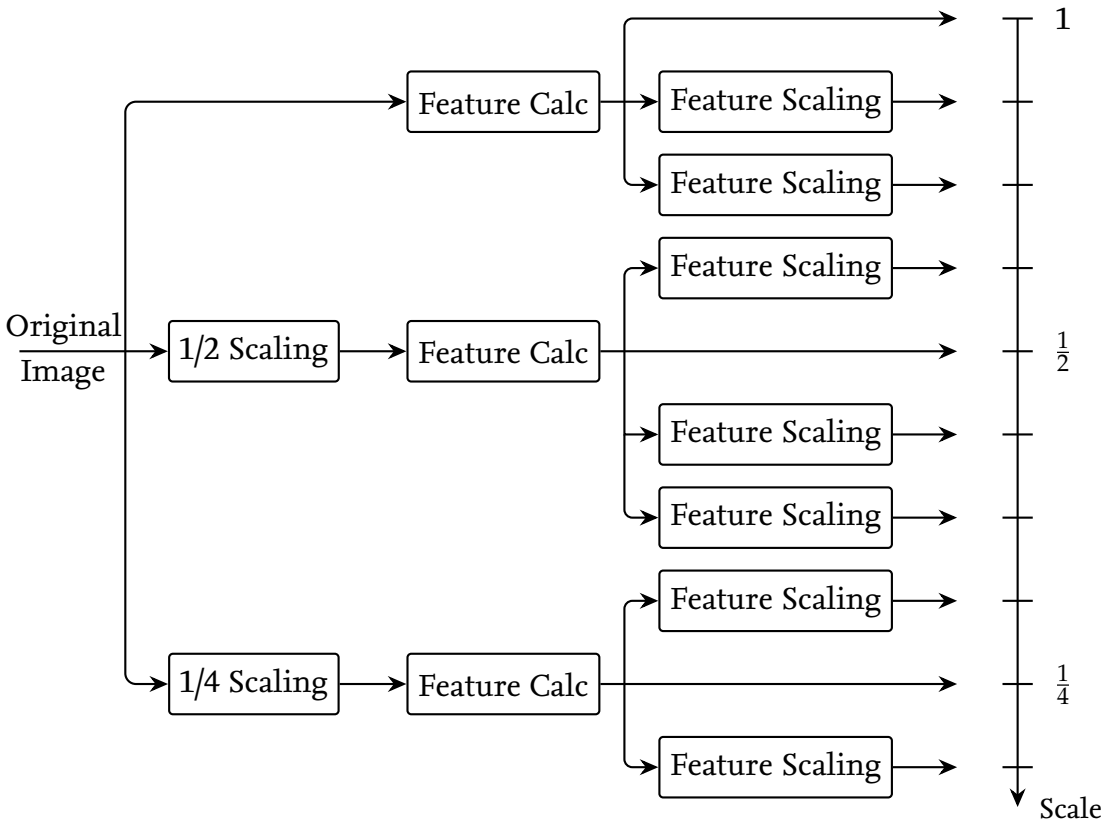


Figure 4.3: Feature extraction by employing the fast feature pyramid approach

- Firstly, for an image at original size, calculate the global mean of each type of channel feature $Mean_{luv_{orig}}$, $Mean_{grad_{orig}}$, and $Mean_{ori_{orig}}$ by:

$$\begin{aligned} Mean_{luv_{orig}} &= \frac{1}{h \times w \times 3} \sum_{k=1}^{3} \sum_{j=1}^{h} \sum_{i=1}^{w} f_{luv}(i,j,k), \\ Mean_{grad_{orig}} &= \frac{1}{h \times w} \sum_{j=1}^{h} \sum_{i=1}^{w} f_{grad}(i,j), \\ Mean_{ori_{orig}} &= \frac{1}{h \times w \times 6} \sum_{k=1}^{6} \sum_{j=1}^{h} \sum_{i=1}^{w} f_{ori}(i,j,k), \end{aligned} \tag{4.4}$$

  where f_{luv}, f_{grad} and f_{ori} are the LUV channel, gradient magnitude channel feature, and histogram of gradient orientation channel images, respectively. h and w are the height and width of the original image. Following Equation (4.4), for each image scale, all global means $Mean_{luv_s}$, $Mean_{grad_s}$, and $Mean_{ori_s}$ can be obtained.

- Then, obtain the proportion matrix $Matrix_p$ from the results of Step 1, which

equals:

$$Matrix_p = \begin{bmatrix} \frac{Mean_{luv_{s1}}}{Mean_{luv_{orig}}} & \cdots & \frac{Mean_{luv_{sn}}}{Mean_{luv_{orig}}} \\ \frac{Mean_{grad_{s1}}}{Mean_{grad_{orig}}} & \cdots & \frac{Mean_{grad_{sn}}}{Mean_{grad_{orig}}} \\ \frac{Mean_{ori_{s1}}}{Mean_{ori_{orig}}} & \cdots & \frac{Mean_{ori_{sn}}}{Mean_{ori_{orig}}} \end{bmatrix}. \tag{4.5}$$

- Finally, calculate the parameter λ_Ω by solving the following equation:

$$\begin{bmatrix} e_1 & \lambda_{luv} \\ e_2 & \lambda_{grad} \\ e_3 & \lambda_{ori} \end{bmatrix} \cdot \begin{bmatrix} 1 & \cdots & 1 \\ \log_2(s_1) & \cdots & \log_2(s_n) \end{bmatrix} = \log_2(Matrix_p), \tag{4.6}$$

  where e_1, e_2, and e_3 indicate the errors between real computing and approximate computing. In practical, the λ_{luv} is 0, and λ_{grad} equals λ_{ori}. Therefore, the parameter λ_Ω is equal to λ_{grad}.

4.1.3 Classification With Boosting Trees

In Section 2.3.2 and 2.3.3, the principle of decision tree and AdaBoost is introduced. In this section, the classification process by employing the AdaBoost with two-depth decision tree is described.

For different object detection applications, due to the difference of detecting objects, the number of decision trees that required are also different. For instance, comparing pedestrian detection and traffic sign detection, pedestrian normally requires more features to describe. Therefore, more features are utilized in pedestrian detection application during the classification process. Correspondingly, more quantity of decision trees is used in pedestrian detection application. In this work, 2048 2-depth decision trees are employed in the pedestrian detection application. Meanwhile, traffic signs which have limited types and colors, are much easier to extract the key features. Therefore, after experimental tests, 400 2-depth decision trees are used.

In the classification process, a sliding window is used for the feature images. Figure 4.4 illustrates the decision-making process of each sliding window. By comparing the feature values and the thresholds from trained classifier data, the confidence value CV_i of each decision tree is calculated and summarized. After that, the final confidence value is compared with a predefined parameter CV_T, which is used to balance the detection accuracy and false positive rate. When the final result is smaller than CV_T, the classification result for current sliding window is

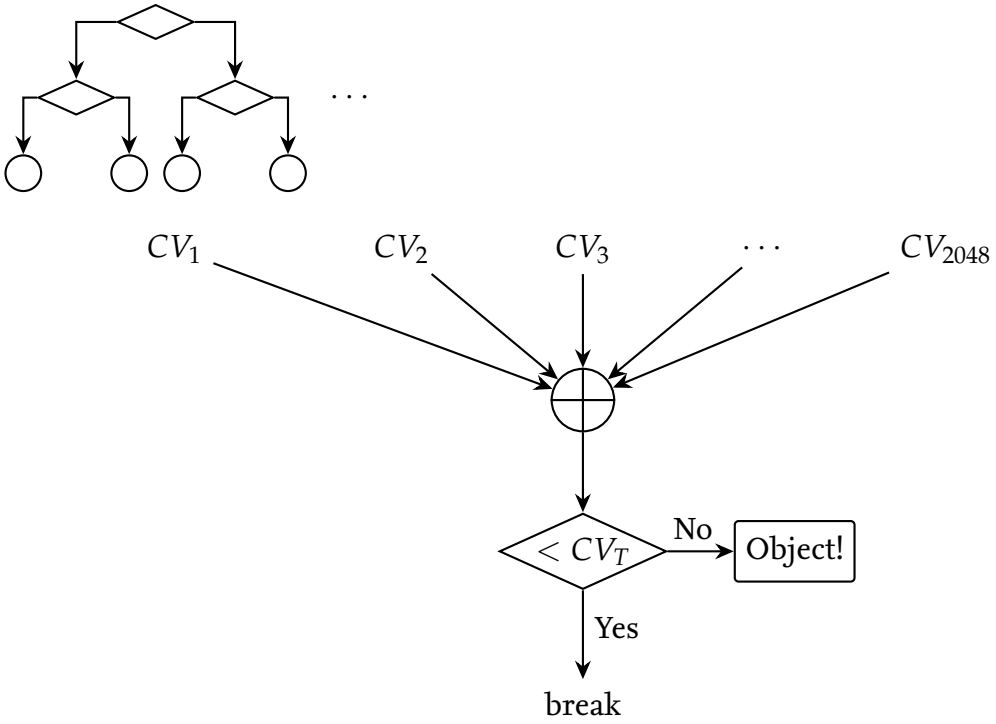


Figure 4.4: An illustration of the classification process with decision trees

non-object. Otherwise, the sliding window is marked as an object and the location information of current sliding window is preserved.

4.2 Framework Validation by Pedestrian Detection Application

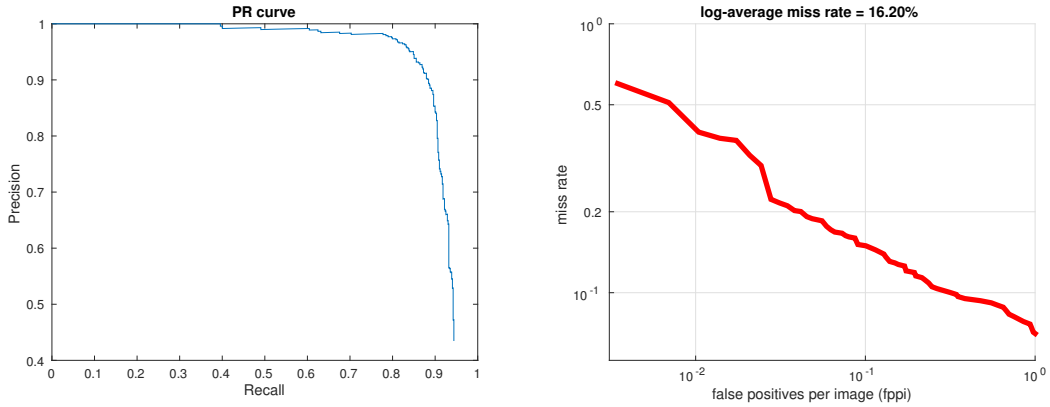
In Section 4.1, the overall structure and key components of the structure are introduced. In this and the following sections, the performance of the generalized object detection framework is tested and verified by different object detection applications. The pedestrian detection algorithm is firstly adapted and verified in this section.

4.2.1 Datasets for Pedestrian Detection

The datasets used in pedestrian detection application are introduced in Section 3.2. In this part, the samples of all three datasets are merged and employed. Since they are already described in Section 3.2, the details of these datasets are omitted here.

4.2.2 Validation Results

The performance of pedestrian detection application by employing the object detection framework is shown in Figure 4.5. In this figure, two performance evaluation methods: Precision-Recall (PR) curve and Receiver Operating Characteristic (ROC) curve are used. The PR curve is a widely used method to evaluate the performance of a classifier. The y-axis "precision" represents how many true positives



(a) PR curve (b) RoC curve

Figure 4.5: The PR curve and RoC curve of pedestrian detection application

are detected among all detected objects, which can be written as:

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}. \tag{4.7}$$

Meanwhile, the x-axis "recall" stands for how many objects are actually detected by the classifier, which can be expressed by:

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}, \tag{4.8}$$

where false negative being the objects not detected by the classifier. Therefore, in PR curve, higher precision means less false positive results in the detection, and higher recall means more percentage of all objects are detected. Hence, for both precision and recall, the higher the better. In other words, the PR curve should be as close to the upper right corner as possible.

The detailed explanation of ROC curve is already introduced in Section 3.4.2. In general, ROC curve is the opposite of PR curve. The ROC curve should be as close to the lower left corner as possible. Because part of the framework structure is based on the ACF algorithm, similar detection performance is obtained after multiple times of training and testing.

4.3 Framework Validation by Traffic Sign Detection Application

With the generalized object detection framework, the traffic sign detection algorithm is adapted and verified in this section. It is worth noting that only traffic sign detection process is performed in the experiments, the traffic sign recognition is not included.

4.3.1 Datasets for Traffic Sign Detection

In the traffic sign detection application, two datasets are employed during the training phase: the German Traffic Sign Detection Benchmark (GTSDB) [68] and the German Traffic Sign Recognition Benchmark (GTSRB) [69]. More than 50,000 annotated images with 42 different traffic signs of Germany are provided by these datasets. Although GTSRB is originally provided for traffic sign recognition, because only the annotated areas are utilized in the positive samples, GTSRB dataset is also suitable for the training phase. Due to that only the traffic sign area is included in the positive samples of GTSRB dataset, GTSRB cannot be used in the testing phase. Therefore, only a number of GTSDB images are employed in the testing phase.

In GTSDB and GTSRB datasets, the traffic signs are categorized into four groups:

- Prohibitory [red circus, inner white background, and black symbols, as illustrated in Figure 4.6]

Figure 4.6: Traffic Sign Category: Prohibitory

- Mandatory [round, inner blue background, and white symbols, as illustrated in Figure 4.7]
- Danger [red triangle, inner white background, and black symbols, as illustrated in Figure 4.8]

Figure 4.7: Traffic Sign Category: Mandatory

Figure 4.8: Traffic Sign Category: Danger

- Others [other symbols, as illustrated in Figure 4.9]

Figure 4.9: Traffic Sign Category: Others

Due to the obvious difference among four groups of traffic signs, each category of traffic signs is trained and tested separately. As a result, four promising classifiers are obtained, where each classifier is used to detect one type of traffic signs. In other words, during the detection phase, the detecting is executed four times for each image, where each time detects one category of traffic signs. Besides, a simplified version of traffic sign detection with only one classifier is also implemented. In this version, all traffic signs are trained together during the training phase. In the end, the performance of two versions of traffic sign detection implementations is compared.

4.3.2 Validation Results of Traffic Sign Detection by Category

Prohibitory Traffic Sign Detection

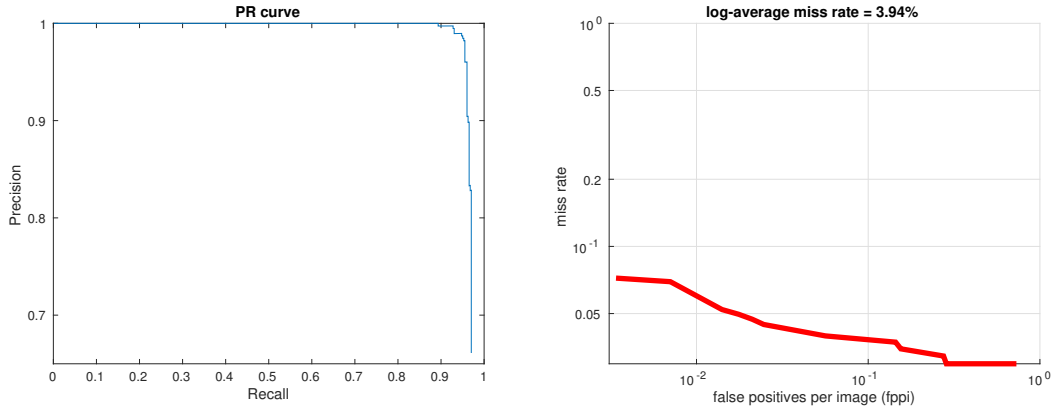
To generate the prohibitory traffic sign classifier, 1918 positive prohibitory traffic signs, whose size is bigger than 56×56, are selected from GTSDB and GTSRB datasets. This amount of positive samples is not enough to train a robust classifier. Considering the traffic signs can be rotated with a maximum 10^o in real-life scenarios, rotating each positive sample by 5^o clockwise and counterclockwise is performed. After the rotation, 5754 positive samples are eventually generated and employed in the training phase of the algorithm.

Regarding negative dataset, no negative image/sample is provided in GTSDB or GTSRB dataset. In this algorithm, the negative dataset is created by manually removing the traffic sign area of GTSDB positive images that used in the training phase. Besides, to make the negative dataset more robust, the negative images of INRIA dataset [22] are also employed as negative samples of this algorithm. In the end, 5822 negative images are finally used in the training phase of prohibitory traffic sign detection.

When the dataset is ready, the classifier is trained and tested with a different number of decision trees. The experimental results show that good performance can be obtained by only employing 100 two-depth decision tree. The PR curve and ROC curve of generated classifier are shown in Figure 4.10. As can be seen, more than 96% of prohibitory signs are detected with the accuracy of ($\backsim$) 97%. (Noting that the classifier showed in Figure 4.10, three stages, [30 60 100], are used during the training phase.)

Considering the number of decision trees that employed in the pedestrian detection application (which is 2048), a number of 100 trees are used in the prohibitory traffic sign detection. The reason that much fewer decision trees are enough can be:

- Firstly, similarities exist among prohibitory traffic signs. All traffic signs are composed by a red circus, white background, and black symbols in the circus, as shown in Figure 4.6.
- Secondly, the pattern of prohibitory traffic signs is finite. All prohibitory traffic signs are shown in Figure 4.6.



(a) PR curve (b) ROC curve

Figure 4.10: The PR curve and ROC curve of prohibitory traffic sign detection

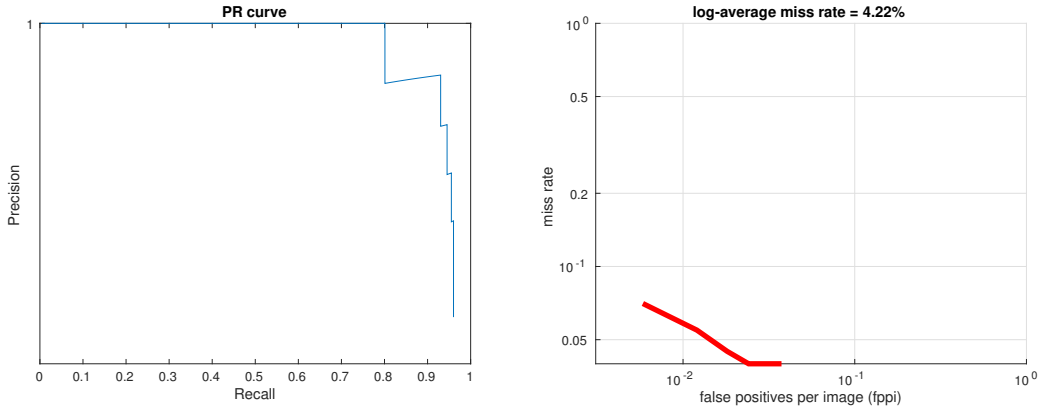
Danger Traffic Sign Detection

In this part, the danger traffic sign classifier is generated. The dataset that employed during the training phase includes 6447 positive samples and 5615 negative images. The positive samples are generated by selecting 2149 positive samples from GTSDB and GTSRB datasets, whose size is bigger than 56×56, and rotating each sample by 5^o clockwise and counterclockwise. The negative dataset of this algorithm is created by a similar strategy of prohibitory traffic sign detection application. Two sets of images are included in the negative dataset: non-traffic sign area of positive images used in the training phase and the negative images of INRIA dataset. For each negative image, multiple negative samples can be obtained by randomly selecting a window of 128×64.

After data training and testing multiple times with different configurations, the performance of final danger traffic sign classifier is shown in Figure 4.11. To generate the illustrated classifier, the configuration of three stages [30 60 120] is used during the training phase. Moreover, the number of decision trees that employed in the final classifier is 120.

Mandatory Traffic Sign Detection

In this subsection, the classifier of detecting mandatory traffic signs is trained. The number of positive samples provided in GTSDB and GTSRB datasets is 808, which is fewer compared with the prohibitory or danger traffic signs. Therefore, rotations with the angle of 3^o, 6^o, 9^o clockwise and counter-clockwise for each sample are performed. For the negative dataset generation, the same strategy of

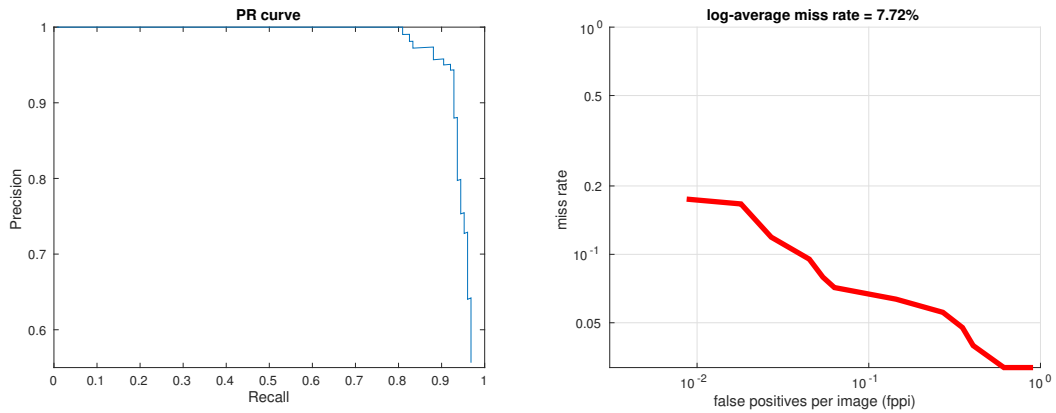


(a) PR curve (b) ROC curve

Figure 4.11: The PR curve and ROC curve of danger traffic sign detection

prohibitory traffic sign detection application is employed. The final dataset that utilized during the training phase of this algorithm includes 5656 ($= 808 \times 7$) positive samples and 4827 negative images.

The performance of generated mandatory traffic sign classifier is shown in Figure 4.12. Although sample rotation is performed to produce enough positive samples, because only 808 original positive samples are provided, the diversity of positive samples is finite. Therefore, the log-average miss rate of the trained classifier is 7.7%, which is slightly worse than the trained prohibitory classifier or the danger classifier. The number of 2-depth decision tree that employed in the final classifier is 100.

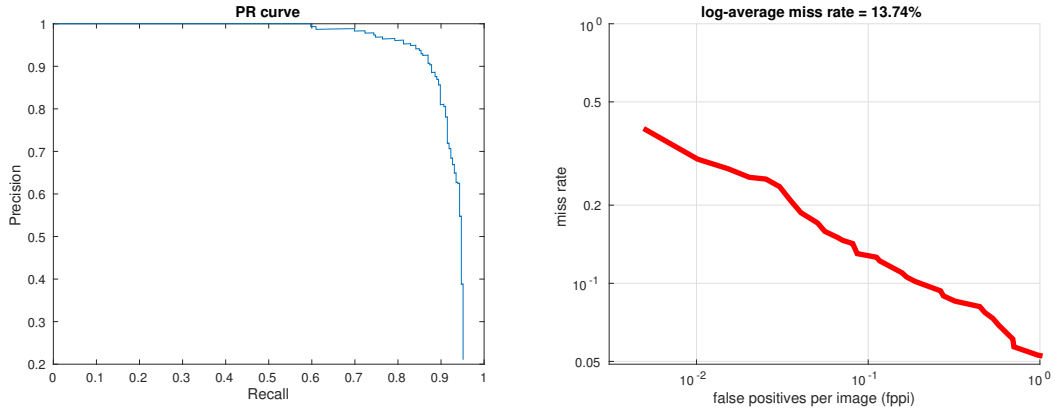


(a) PR curve (b) ROC curve

Figure 4.12: The PR curve and ROC curve of mandatory traffic sign detection

Others Traffic Sign Detection

The traffic signs that can not be categorized into prohibitory, danger, or mandatory, which are shown in Figure 4.9, comprise the "Others" category of traffic signs. The classifier of these traffic signs is trained in this part. To generate the classifier, 1308 positive samples are selected from GTSDB and GTSRB datasets. By rotating each positive sample with the degree of 5^o and 9^o, 6540 positive samples are created. As for the negative dataset, 4843 negative images are employed in the data training. After training and testing with different parameters multiple times, the performance of the final classifier is shown in Figure 4.14.



(a) PR curve (b) ROC curve

Figure 4.13: The PR curve and ROC curve of others traffic sign detection

As can be seen from Figure 4.14, the miss rate of detecting this category of traffic signs is relatively high. Referring to Figure 4.9 listed traffic signs of this group, we can notice that fewer similarities exist among the traffic signs of "Others" category. Therefore, inevitably, relatively worse performance is obtained by this classifier compared with the classifiers of danger, mandatory, and prohibitory traffic signs.

Combining Four Detectors

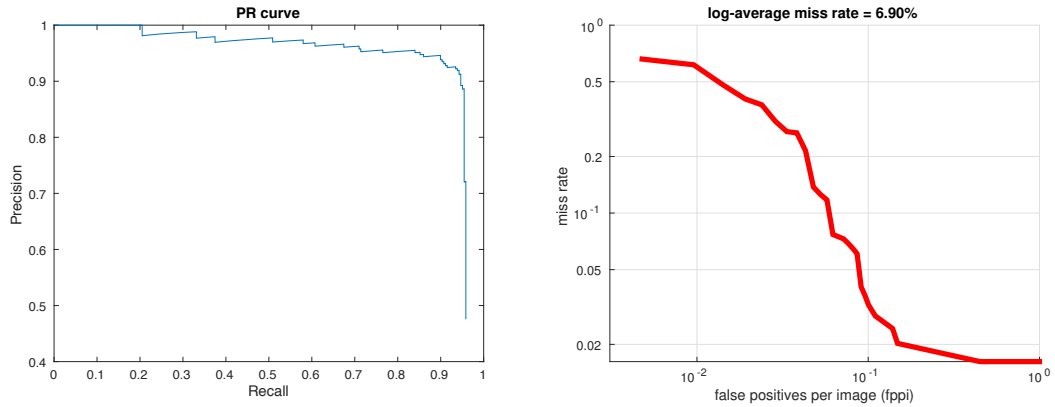
After four types of traffic sign classifiers are trained, the final detector is obtained by combining all four classifiers together. Hence, during the classification process, for each sliding window, four times of detection are performed where each time detects one category of traffic signs. The number of decision trees that utilized in the classifiers is 100, 120, 100, and 300, respectively. Compared with 2048 trees used in pedestrian detection, the number of 620 trees used in traffic sign detection algorithm is much less. Therefore, in practical, the detection speed for

each sliding window of traffic sign detection is 2 ∽ 3 times faster than the pedestrian detection application.

4.3.3 Validation Results of Traffic Sign Detection All together

In Section 4.3.2, the traffic sign detection is achieved by training and employing four classifiers that each classifier detects one category of traffic signs. In this section, a simplified version of traffic sign classifier, which generated by training all types of traffic signs together, is introduced.

In GTSDB and GTSRB datasets, the number of positive samples is 6183. Employing non-traffic sign area as the negative images plus negative images of INRIA dataset, 5813 negative images are prepared. After classifier training with different number of decision trees, different number of positive samples (by rotating positive samples), different number of training stages, the results show that training with four stages, [50 100 200 400], achieves relative good performance, which is shown in Figure 4.14. Moreover, the experiments also reveal that the performance of the classifier is not improved by using more rotated positive samples.



(a) PR curve (b) ROC curve

Figure 4.14: The PR curve and ROC curve of traffic signs trained all together

As shown in Figure 4.14, 97 percent of traffic signs is detected by this version of algorithm and the log-average false positive rate is 6.9 percent. This result is good enough compared with the traffic sign detection by category approach. However, after analyzing the training dataset of this algorithm, we discover that much less number of mandatory and others category traffic signs are provided in the dataset. Therefore, in practical, the classifier of "traffic sign detection all together" is weaker while detecting the signs of mandatory and others category.

4.4 Framework Validation by Head Detection Application

Head detection is an application similar to face detection. To simplify the annotation work of faces, human heads are annotated and detected in the thesis. Therefore, the generalized object detection framework is adapted and verified by the head detection algorithm in this section. In the following, the dataset employed and the validation results of the head detection algorithm are explained.

4.4.1 Dataset for Head Detection

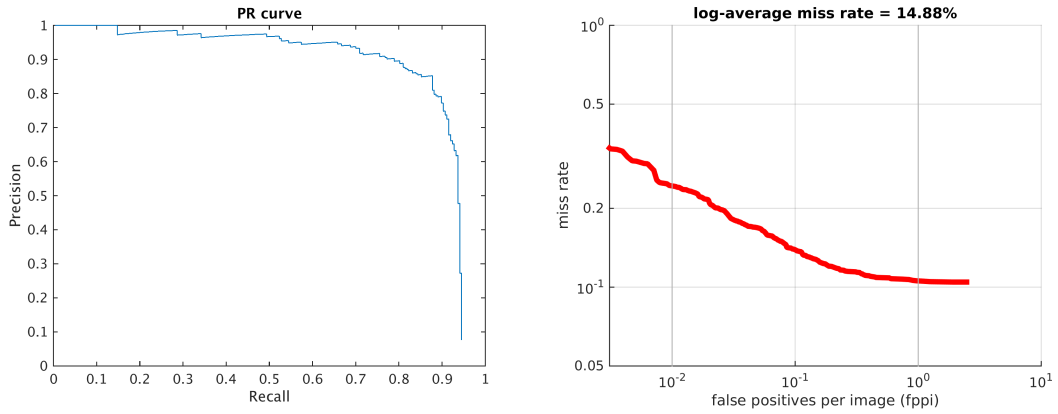
Face detection is a popularized algorithm in computer vision and robotics fields. Nevertheless, the annotation methods of the face datasets available online are different from pedestrian/traffic sign datasets. In pedestrian/traffic sign datasets, objects are annotated by rectangles. However, most of the face datasets are annotated either in the shape of an eclipse, or the position of eyes, noses, etc.

In this work, Caltech-Head dataset [70] is employed. Owing to the annotation of the head in Caltech-Head dataset is in the shape of eclipse, a conversion of annotations from eclipse to rectangle is performed with MATLAB. Besides, a set of face samples from "WIDER FACE" [71] dataset is also utilized during the training phase. In "WIDER FACE" dataset, the faces are annotated with expressions, poses, illuminations, occlusions, etc. To simply the detection system, only the samples of front-face are selected and used.

4.4.2 Validation Results

In this part, 4366 positive samples are employed. Similar to traffic sign detection application, the non-face areas of each positive image plus the negative images of INRIA composed the negative dataset. After data training and testing with different configurations multiple times, the performance of final face classifier is shown in Figure 4.15.

As can be seen from Figure 4.15, the log-average miss rate of head detection is 14.88%, which is acceptable compared with other face detection algorithms [25], [53]. However, since only front-face/head samples are employed in classifier training, only front-face/head can be detected.



(a) PR curve (b) RoC curve

Figure 4.15: The PR curve and RoC curve of head detection application

4.5 Summary

In this chapter, a generalized object detection framework is created. Thereafter, the configurable parameters of this framework are explained by introducing the theory of "fast feature calculation" approach. With this framework, different detection applications are programmed and verified. Finally, the performance of each detection application is given and explained.

Chapter 5

IP-Toolbox Design for the Generalized Object Detection System

Based on the generalized object detection framework introduced in Section 4.1, IP cores that correspond to each module of the object detection algorithm are designed. In this chapter, the design process and details of each IP core are introduced.

5.1 Camera Capture

In this thesis, the research interest is focused on the object detection algorithms / systems that utilizing images captured from normal video cameras as the system input. Therefore, the camera model and the image data transfer interface are compared firstly. (The details about camera selection will be introduced in Section 6.1.) After comparison, Pmod camera OV7670 [72], which is widely used in open-source embedded projects, is deployed as the system camera.

Mike Field [73] provides a basic open source project that utilizing OV7670 and Zedboard. In this work, to obtain better images from this camera, further camera configuration, calibration and image processing procedures are performed. The overall structure of camera capture module is presented in Figure 5.1.

From Figure 5.1, it is evident that this module consists of 5 submodules. The function and details of each submodule are introduced in the following parts of this section.

5.1.1 Camera Config Generator

The task of the camera config generator submodule is to provide valid and correct camera configuration register addresses and values sequentially. These configuration data indicate the frequency, resolution, etc. information of OV7670 camera.

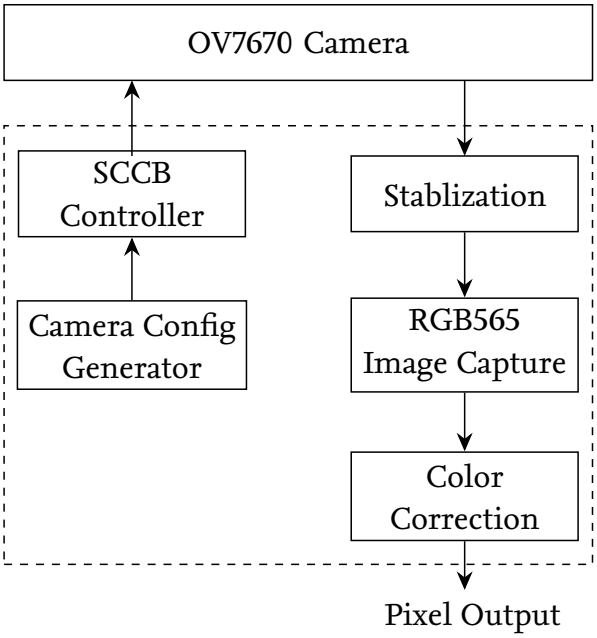


Figure 5.1: The overall structure of OV7670 camera capture module

In this thesis, the OV7670 camera is always configured as Table 5.1 shows. In this table, RGB565 is the camera output data format, which uses 5 bits for red channel, 6 bits for green channel, and 5 bits for blue channel in each pixel.

	Clock	FPS	Resolution	Data Format
Configuration	24Mhz	30	640×480	RGB565

Table 5.1: The employed configuration of OV7670 camera

5.1.2 SCCB Controller

Serial Camera Control Bus (SCCB), which is defined and deployed by OmniVision Technologies [74], is an I^2C-like protocol that used for the configuration of OV series cameras. For OV7670 Pmod camera, 2-wire SCCB interface is used, although 2-wire SCCB interface only allows the master connecting to one slave device. Here the details of SCCB transmission protocol are not described since it is out of the topic of this thesis. More information about SCCB is available from its specification [75]. In general, the SCCB controller submodule serves as the interface between camera configuration data and the camera. Therefore, all camera registers are configured via SCCB.

5.1.3 Stabilization

From Section 5.1.1, we know that the OV7670 camera is configured with the RGB565 data format. Therefore, the corresponding image data receiving submodule should also match this pattern. Due to the limitation of camera pins, only 10-bit data could be outputted from the camera per clock cycle. Namely, 8-bit pixel data and two one-bit signals: "vsync" and "href", which indicate the image frame synchronization and line break, respectively.

In reality, because of the camera quality and data transmission noise, camera data are not firmly generated per clock cycle. Therefore, a FIFO-based stabilization submodule is required to stabilize the output of the camera. In the design, "ov7670 _pclk" is used as the input clock of the FIFO and the clock rate of data output after stabilization is 25MHz. Through on-board testing with an oscilloscope for many times, the results show that 256-depth FIFO is enough to stabilize the camera outputs. The FIFO width is set to 10 bits (8-bit output data, href and vsync).

5.1.4 RGB565 Image Capture

After the stabilization, the image data are firmly generated every clock cycle. However, for each pixel of a color image, RGB565 contains 16 bits. Therefore, the data of each pixel is obtained by two clock cycles, as illustrated in Figure 5.2. In order to simplify the data format for future image processing modules, 16-bit RGB is generated by combining 8-bit camera output every two clock cycles. After that, optionally, 16-bit RGB data can be reformed to 24-bit RGB by zero padding for each color channel.

clock 1: First byte	R4 R3 R2 R1 R0	G5 G4 G3
clock 2: Second byte	G2 G1 G0	B4 B3 B2 B1 B0

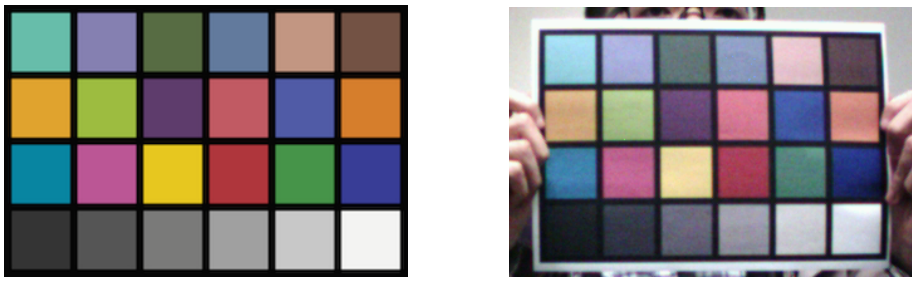
Figure 5.2: The format of RGB565 data output

5.1.5 Color Correction

Color correction is an image preprocessing procedure that calibrates the color of captured pictures to the color of real world. For many vision applications, it is optional. However, the task of this work is to detect different objects, which may contain features that are sensitive to colors. Moreover, the employed OV7670 camera produces images with obvious color distortion. Therefore, color correction

submodule is designed to enhance the picture quality captured by the camera.

To achieve the color correction, Colorchecker [76] is used, as shown in Figure 5.3(a). The colors on colorchecker are predefined in its specification document [77]. Thus, firstly, some pictures of Colorchecker are captured by OV7670 camera



(a) Standard Colorchecker

(b) ColorChecker captured by OV7670

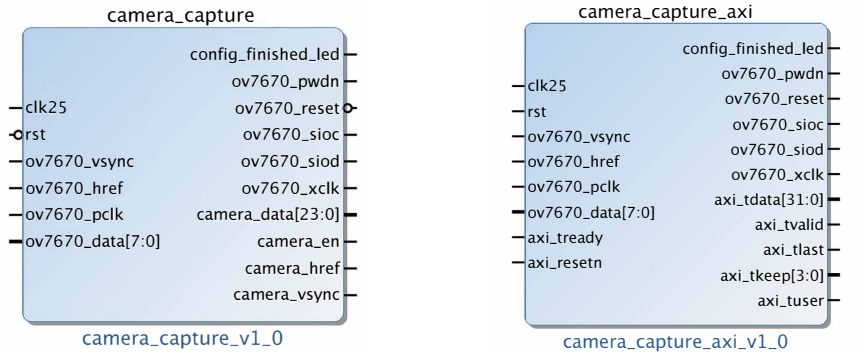
Figure 5.3: A comparion between standard Colorchecker and the image captured by OV7670 camera.

under good light conditions, as shown in Figure 5.3(b). Then, the average value of each color on Colorchecker is calculated by Matlab. Finally, the polynomial-based color correction [78] algorithm is executed by comparing predefined color values to the color values that Matlab calculated. The resulting vector serves as the color matrix that converts colors of captured image to real world color. With this color conversion matrix, all pixels obtained from RGB565 Image Capture submodule can be color-corrected.

IP Packaging and Resource Utilization Analysis

In this work, to give the flexibility to further project design with camera capture IP, two versions of camera capture IP are generated: camera capture without AXI interface version and with AXI interface version, as shown in Figure 5.4. Without AXI interface, this IP can be directly connected to other image processing IPs, such as undistortion, rectification or image filtering. On the other hand, with the AXI interface, the image data could be transferred to the DDR3 memory easily by Video Direct Memory Access (VDMA).

In Figure 5.4, “ov7670_∗” are the ports that interface this IP with OV7670 camera. “config_finished_led” is the signal that indicates whether the OV7670 configuration is finished. “camera_*” are the final stable and color corrected output from the no AXI interface version IP. “camera_href” and “camera_vsync” indicate the frame synchronization and line breaks of output image. “camera_data” and “camera_en” are the data output and valid signals. In Figure 5.4(b), the“axi_∗” ports are



(a) Camera capture IP without AXI

(b) Camera capture IP with AXI

Figure 5.4: The packaged IP of camera capture module

the standard AXI interface. 24-bit RGB pixel data is extended to 32bits for easier transmission and storage in DDR3 memory. The "camera_href", "camera_vsync", and "camera_en" signals are converted to AXI version signals "axi_tlast", "axi_tuser", and "axi_tvalid", respectively.

The resource utilization of two IP cores on Zedboard is shown in Table 5.2. In both IPs, three DSPs are consumed by color correction submodule. Besides, similar number of resources are used by the two IP cores.

Site Type	Resource Used		Available
	IP Without AXI	IP With AXI	
LUT	199	203	53200
FF	262	269	106400
BRAM	0	0	140
DSP	3	3	220

Table 5.2: The resource utilization report of camera capture IP on Zedboard

5.2 StereoCam Capture

In Section 5.1, capture IP by single camera is designed. For some detection applications, stereo images are required as the data input. Therefore, StereoCam capture module, which can interface with two Pmod cameras, is introduced in this section. (The details of StereoCam board will be explained in Chapter 6.)

The overall structure of StereoCam capture module is shown in Figure 5.5. Because two OV7670 cameras are used in this module, two camera config generators

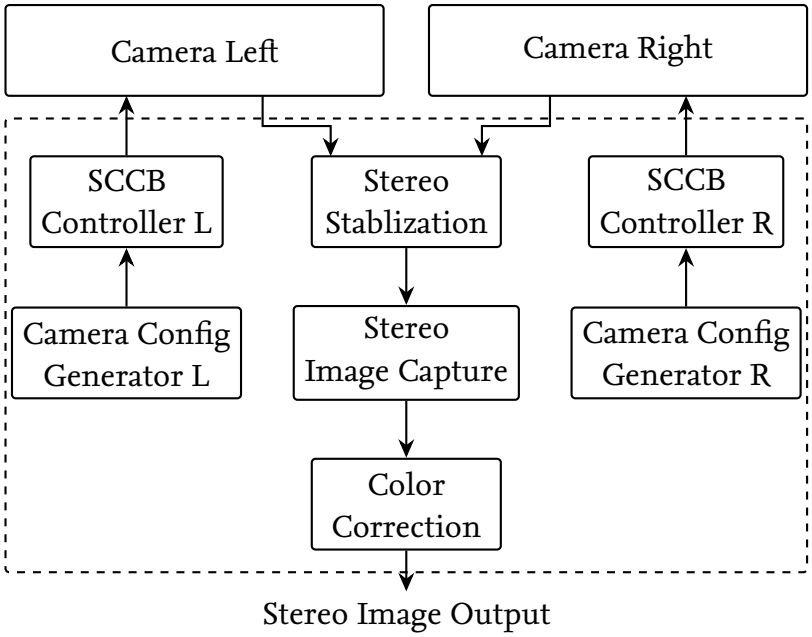


Figure 5.5: The structure of StereoCam capture module

and two SCCB controllers are required. Besides, the revised version of stabilization and image capture submodule are implemented to handle the data from two cameras.

Since each block in Figure 5.5 is similar to the submodules inside single camera capture IP, the detailed introduction for each submodule is omitted. The view of packaged IP and resource utilization are shown in Figure 5.6 and Table 5.3, respectively.

To obtain the stereo information (e.g., depth) from stereo cameras, the calibration and rectification of stereo images are always required. Thus, AXI interface is not necessary for this module. Besides, in later Section 5.3 StereoCam rectification module, many lines of pixels should be buffered for both images, therefore, the data output of two cameras: "cam1_out_data" and "cam2_out_data" are 16 bits instead of 24 bits, which can save the usage of BRAM resource in rectification module.

Comparing the resources used by StereoCam capture module in Table 5.3 and single camera capture IP used in Table 5.2, we can conclude that the resources consumed by StereoCam capture IP is around doubled of the single camera capture IP. However, because only one stabilization and one image capture module is used, therefore, slightly less than twice the size of resources are used by StereoCam capture IP.

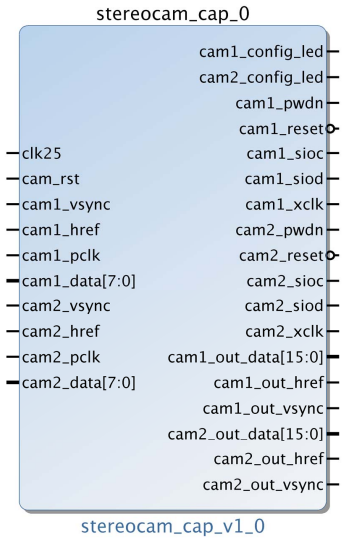


Figure 5.6: The packaged IP of StereoCam capture

Site Type	Resource Used	Available
LUT	389	53200
FF	506	106400
BRAM	0	140
DSP	6	220

Table 5.3: The resource utilization report of StereoCam capture IP on Zedboard

5.3 StereoCam Rectification

StereoCam rectification is the process of reprojecting the image planes of two cameras to achieve co-planar and row-aligned stereo images. Figure 5.7 illustrates a pair of images captured by my stereo cameras. The red lines in this figure are added afterwards to show the status of alignment between two images. From these red lines, it is evident that left and right image are not aligned. Besides, some distortions in the corners of both images can also be observed from this figure.

Without image rectification, disparity/depth information cannot be obtained from stereo images. Therefore, image rectification is a necessary and critical step for any stereo camera-based applications. And the quality of rectification directly concerns the accuracy of disparity/depth information that calculated from stereo images.

In order to accomplish image rectification, the following steps should be performed:

- Firstly, stereo camera synchronization, which guarantees two cameras are capturing and storing the same frame of the image.
- Secondly, stereo camera parameters calculation, including the intrinsic matrix for each camera, camera distortion coefficients, and the extrinsic matrix. Distortion coefficients are used to eliminate camera distortion. Intrinsic

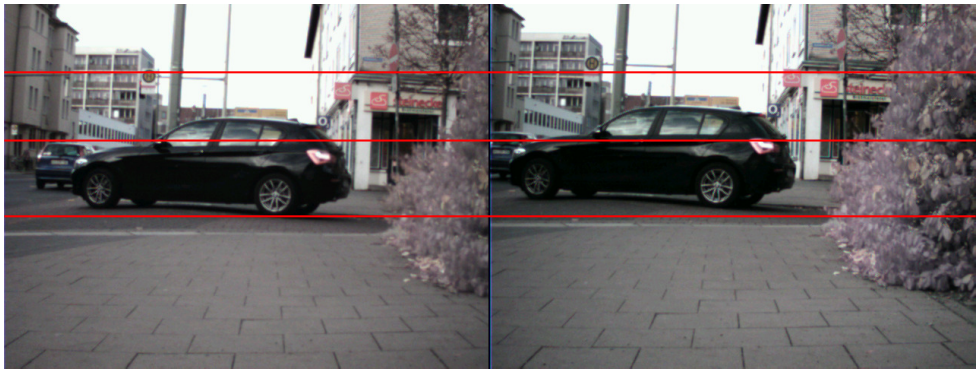


Figure 5.7: An example of stereo images that captured by two OV7670 cameras: left and right images are not aligned and distortion exists in both images

and extrinsic matrix are used to reproject image planes to co-planar.

- Finally, calibration and rectification with stereo camera parameters.

5.3.1 StereoCam Synchronization

Although two cameras are configured at exactly the same time by the same reset trigger and using the same clock that provided by FPGA, due to the manufacturing and wiring problem, slightly differences, such as the delay before first data output, still exist between two cameras in reality. Therefore, camera synchronization is required for further image capturing.

To figure out the timing difference between two cameras, the "OV7670_vsync" signal from two cameras are scoped with an oscilloscope. After camera resetting and startup multiple times, the experimental results show that usually one camera output is delayed or ahead of the other camera output $0 \sim 700$ pixels. Therefore, 1024-depth FIFO for each camera is used in this submodule.

Figure 5.8 illustrates an example of the timing difference between two cameras' data output. Assuming at Time A, the first data of left camera is outputted. Then, the data from left camera is stored into its FIFO every clock cycle. At Time B, 402 clock cycles later, the first data of right camera is generated and sent to FIFO. For the reading process, at Time C (this 100 clock cycles delay is optional), "Data1"s from the left and right camera FIFO are read out. Thereafter, this reading process from FIFO continues for each clock cycle. In this submodule, the data that passed through FIFO contains 16-bit RGB565 pixel data, "camera_href" and "camera_vsync" signals.

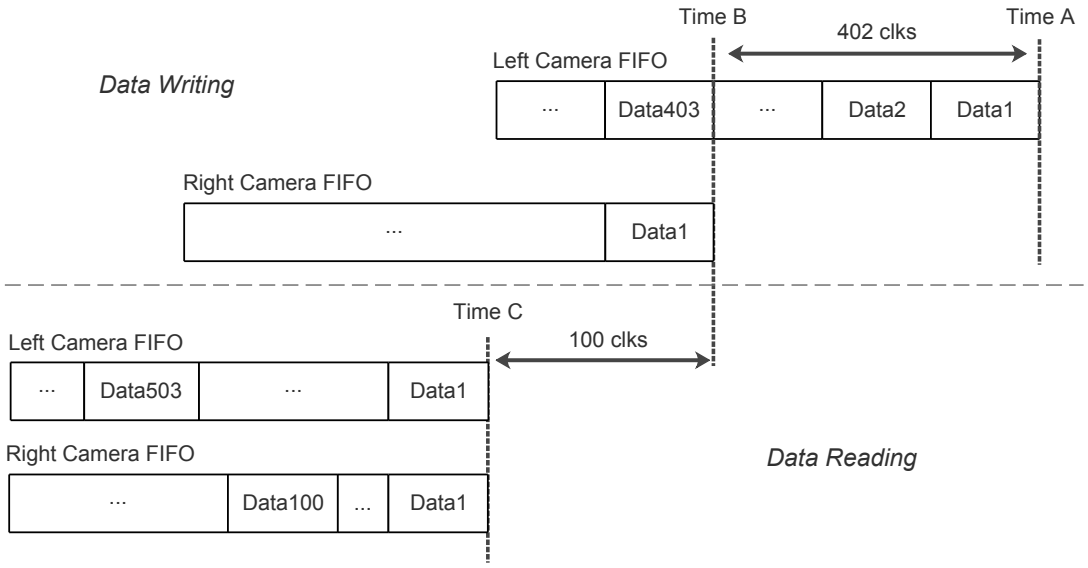


Figure 5.8: An example of StereoCam synchronization process: FIFO writing with time difference and FIFO reading at the same time

5.3.2 StereoCam Parameter Calculation

As mentioned in Section 5.3, two problems exist in the stereo images that captured by the two OV7670 cameras: image distortion and non-coplanar. Therefore, correspondingly, two sets of parameters are required to eliminate these problems: distortion coefficients of two cameras; intrinsic matrix of each camera and extrinsic matrix.

Image Distortion

It is known that the camera imaging principle is based on the pinhole camera model. However, in real cameras, lens are employed to replace the ideal but unavailable pinhole. Due to the manufacturing technology limits, non-alignment between lens and imager, "spherical" lens, etc. problems exist [79]. These problems result in the tangential distortion and radial distortion of the camera, respectively.

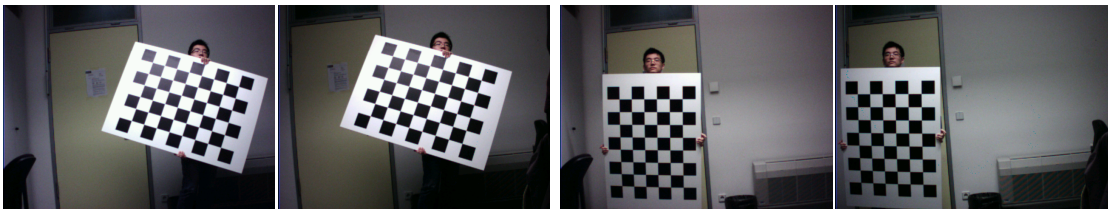
In order to remove these distortions, a series of “checkerboard” photos are required to capture by the camera, as depicted in Figure 5.9. Thereafter, the calibration is processed by Zhang’s method [80]. The calibration results, intrinsic matrix and distortion coefficients, are the parameters required for further rectification or undistortion.

Figure 5.9: An illustration of OV7670 captured checkerboard images

Non-coplanar

After introducing lens distortion which exists in all camera, this part will talk more about the other problem that exists between stereo cameras: non-planar or non-coplanar. In practical, when installing two cameras, although we tried to pose them parallel, two cameras are neither horizontally aligned nor coplanar, which are actually two necessary requirements for stereo imaging model.

To solve this problem, stereo camera calibration is executed. In the stereo calibration process, intrinsic matrix and distortion coefficients are used as the known input. A series of stereo "checkerboard" photos is captured, as shown in Figure 5.10. Then Bouquet's Matlab Toolbox [81] is employed to calculate the extrinsic matrix, which represents the relationship between two cameras.



(a) Left (b) Right (c) Left (d) Right

Figure 5.10: An illustration of StereoCam captured stereo checkerboard images

Until now, all the parameters required by stereo camera rectification are computed. Noticing that the intrinsic matrix and distortion coefficient are related to the change of camera focal length. Although OV7670 camera contains a manually rotatable lens, once camera parameters are calculated, the camera lens should never be rotated. Otherwise, both single camera calibration and stereo calibration should be repeated. Moreover, the relative position of two cameras should never be changed either. Otherwise, the extrinsic matrix would be invalid. Besides, the process of StereoCam parameter calculation is not inside StereoCam rectification IP. This calculation is only required to be done once with Matlab or OpenCV on

a PC.

5.3.3 Stereo Image Rectification

In Section 5.3.2, camera intrinsic matrix, distortion coefficients, and extrinsic matrix are obtained. With these parameters, the stereo images could be rectified by two steps: homography transformation and undistortion.

Homography Transformation

Homography transformation is the process of a coordinate transformation from camera coordinate system to the new coplanar coordinate system. This transformation guarantees that stereo images are coplanar and row aligned after the conversion. During this process, homography transformation matrix, which can be calculated from the intrinsic and extrinsic matrix, is required. (More information about homography transformation theory can be found in [79].)

Undistortion

Undistortion is the process of removing image distortion with distortion coefficients. As mentioned in Section 5.3.2, distortion contains radial distortion and tangential distortion. After the camera calibration with "checkerboard" pictures, the Matlab results show that tangential distortion of OV7670 camera that used in the system is tiny. Therefore, only radial distortion is considered and removed in this IP.

A standard approach to model the radial distortion is using Taylor series expansion around r, which is the distance between the pixel in the image and the principal point. Then, the transformation is processed according to

$$\begin{aligned} x_{dist} &= x_{undist}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{dist} &= y_{undist}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{5.1}$$

where (x_{undist}, y_{undist}) being the undistorted point that projected by an ideal pinhole camera and (x_{dist}, y_{dist}) being the projected distorted point in reality. Besides, it is known that $r^2 = x_{undist}^2 + y_{undist}^2$. From Equation (5.1), we can defer that the distortion at imager center (optical axis intersect) is 0; the distortion increases as the point move towards the periphery/margin. And the distortion can be expressed by 3 parameters k_1, k_2, and k_3. These parameters actually are the distortion coefficients obtained from offline parameter calculation process of Section 5.3.2.

In this IP design, to achieve stereo image rectification, homography transformation and undistortion process are required to execute for all image pixels. As illustrated in Figure 5.11, the original left image $I_l''(x_l'', y_l'')$ and right image $I_r''(x_r'', y_r'')$ are distorted and non-coplanar. After undistortion processing, the images are converted to undistorted images $I_l'(x_l', y_l')$ and $I_r'(x_r', y_r')$. Then, rectification is performed via homography transformation matrix to the undistorted images and the resulting images $I_l(x_l, y_l)$ and $I_r(x_r, y_r)$ are row-aligned and coplanar images.

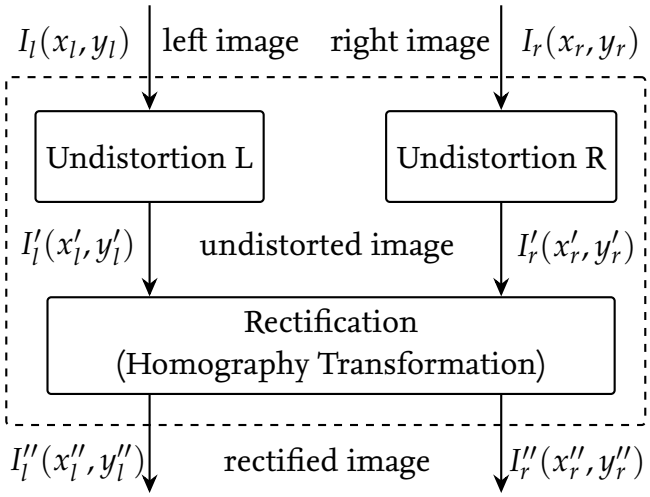


Figure 5.11: The process of image rectification

However, in the implementation, to guarantee all points in the new coplanar coordinate system are not void, the reverse process is performed. The practical calculation process can be described as follows:

- Step 1: For each point of $I_l(x_l, y_l)$ and $I_r(x_r, y_r)$ in the new coplanar coordinate system, the corresponding points $I_l'(x_l', y_l')$ and $I_r'(x_r', y_r')$ in the undistorted images are calculated. During this process, homography transformation using the inverse homography transformation matrix H_l^{-1} or H_r^{-1} is performed.

- Step 2: Locate the corresponding points $I_l''(x_l'', y_l'')$ and $I_r''(x_r'', y_r'')$ in the distorted image, namely, OV7670 camera captured image, by Equation (5.1).

- Step 3: Check if the x, y coordinates of (x_l'', y_l'') and (x_r'', y_r'') are in the range of $0 \sim 639$ and $0 \sim 479$, respectively. Pixel values of all points out of this range are set to 0.

- Finally, this process is repeated for each pixel until all pixels in the new

coplanar image are processed.

IP Packaging and Resource Utilization Analysis

The packaged IP of StereoCam rectification module is shown in Figure 5.12. "clk25" and "clk100" represent two clock input 25MHz and 100MHz, respectively. "clk25" serves as the input clock of image pixel. One 16-bit pixel is read into this module every two cycles of "clk25" clock. However, for each pixel, the rectification process requires many cycles to finish. In order to make the real time rectification possible, four-stage pipeline structure is employed. Each stage consumes maximum eight cycles. To make the pipelines time-aligned and real-time, "clk100" is used as the clock input of rectification pipeline block and the state machine of each stage contains eight states.

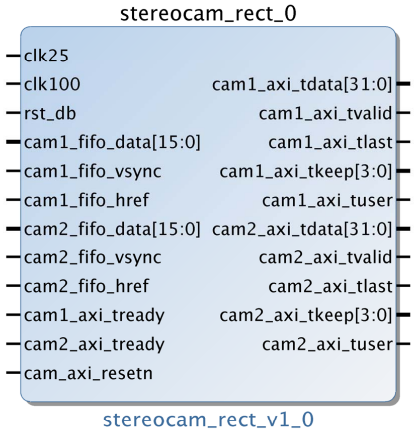


Figure 5.12: The packaged IP of StereoCam rectification module

The resource utilization of StereoCam rectification IP is listed in Table 5.4. In this module, in order to guarantee that the corresponding pixels are always available, 64 lines of pixels are buffered for both left and right image, where each pixel is 16-bit in the RGB565 format. Therefore, 39 BRAMs are utilized in Table 5.4. Moreover, due to the computation complexity of homography transformation and undistortion process, a reasonable number of DSPs is also consumed in this module.

Finally, an example pair of rectified stereo image result is shown in Figure 5.13. (The original captured images are also shown in Figure 5.7.) The red lines in this Figure are drawn afterwards to show the alignments between two images. Comparing the objects/colors/pixels between left and right images on each red line, it is evident that after rectification, two images are almost perfectly aligned. The only difference between left and right images is the pixel displacement in the X-axis direction, which should exist and is extremely critical for further depth calculation.

Site Type	Resource Used	Available
LUT	3661	53200
FF	8706	106400
BRAM	39	140
DSP	44	220

Table 5.4: The resource utilization report of StereoCam rectification IP on Zedboard

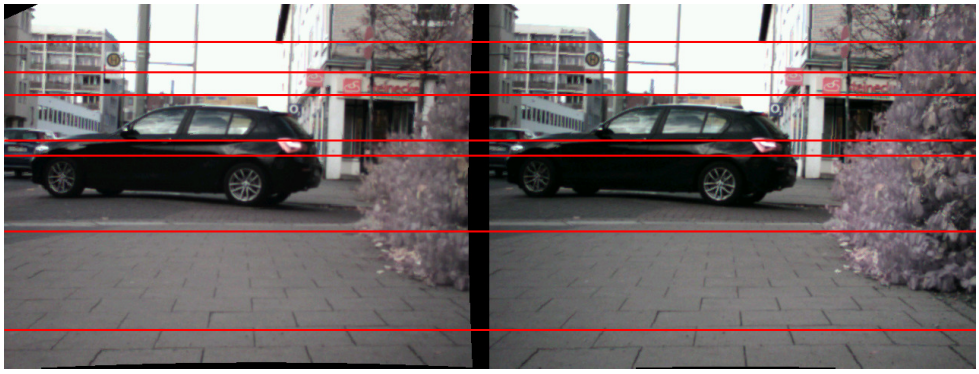


Figure 5.13: An example of rectified stereo images captured by two OV7670 cameras: left and right images are row-aligned and coplanar

5.4 Depth Calculation

Referring to Figure 4.1, depth calculation module accepts the rectified stereo images and generates the disparity image for each pair of inputs. Based on the disparity information, the depth/distance of an object to the camera can be further obtained. The geometry of the depth estimation from stereo images is illustrated in Figure 5.14 [79].

In Figure 5.14, the left and right cameras are assumed to be aligned and undistorted; the optical axes are parallel. The projection center of each camera is denoted as O_l and O_r, respectively. Object S and its projected points in left and right images are represented as $S_l(x, y)$ and $S_r(x', y)$, respectively. The disparity d at point S_l, can be obtained by

$$d = x - x'. \tag{5.2}$$

If we denote the focal length as f, the length of baseline $O_l O_r$ as B, and the distance

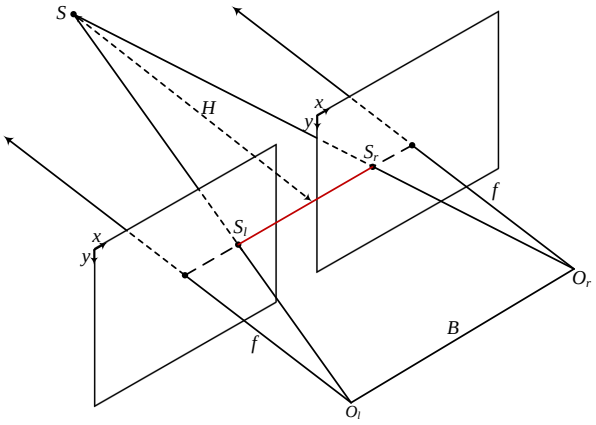


Figure 5.14: Geometry of depth estimation from stereo cameras

of the object S to camera as H, we have

$$\frac{H}{H+f} = \frac{B-d}{B}, \tag{5.3}$$

which results in

$$H = \frac{fB}{d} - f. \tag{5.4}$$

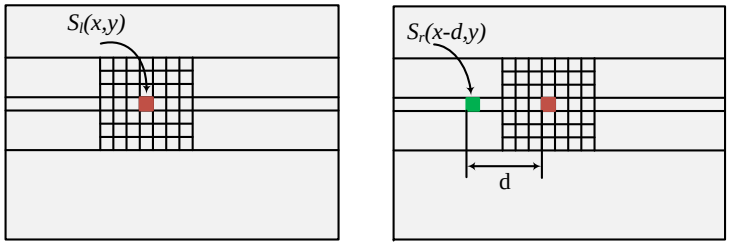
Thereafter, the depth Z of object S equals

$$Z = H + f = \frac{fB}{d}, \tag{5.5}$$

with both focal length f and baseline length B being fixed and known. According to Equation (5.5), the depth Z is only related and inversely proportional to the disparity d, which is reasonable since closer objects have bigger disparity [79]. Therefore, depth calculation is actually the process of finding the disparity of corresponding pixels from stereo images. In other words, depth estimation is a stereo matching process for each pixel.

In this module, Sum of Absolute Difference (SAD)-based stereo matching is employed. Figure 5.15 exemplifies the stereo matching process for object point S. For the ease of description, we re-denote the projected points of S on two images as $S_l(x,y)$ and $S_r(x-d,y)$ in this part. If we use a $N \times N$ floating window with S_l and S_r as the center on each image, respectively, little differences should exist between two window blocks. To measure the intensity differences, SAD is calculated and denoted as T. For arbitrary point $S_l(x,y)$ in Figure 5.15(a) and its potential corresponding point $S_r(x-d,y)$, SAD of all pixels inside floating window can be computed by

$$T = \sum_{(i,j)\in N} \left| S_l(x+i,y+j) - S_r(x-d+i,y+j) \right|. \tag{5.6}$$



(a) Arbitrary point S_l in left image

(b) Corresponding point S_r in right image

Figure 5.15: An illustration of SAD-based stereo matching process

Therefore, if pixel $S_l(x,y)$ position in left image is given, the process of locating $S_r(x-d,y)$ on the right image is a block matching approach. Since the matching pixel in right image should be on the left side of $S_l(x,y)$, one pixel left shifting of the floating window is executed and the SAD is calculated for each disparity. Assuming the maximum disparity as M, after M times shifting and computing, the minimum SAD corresponding disparity is selected as the disparity at $S_l(x,y)$. Repeating this process for every pixel until the disparity map of the whole image is obtained.

In the design, to save the computing resources and the time of data accessing, column SAD (col-SAD) is calculated and buffered, as illustrated in Figure 5.16.

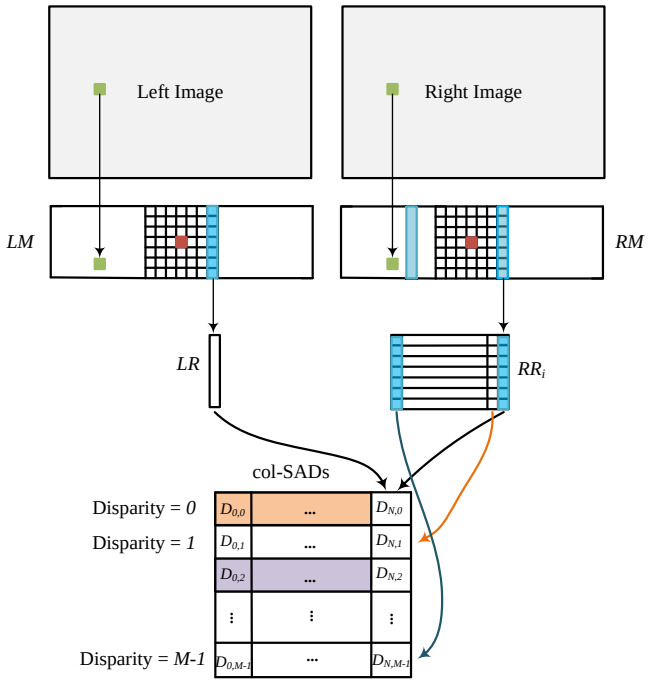


Figure 5.16: An illustration of col-SAD-based stereo matching process

To realize real-time processing of multiple data, LM and RM are employed to buffer the input pixels. Moreover, a window-column size local memory LR is used to store left image pixels. An M columns memory RR_i is used to store right image pixels, with $i = \{0, 1, \ldots, M-1\}$. An $M \times (N+1)$ memory block col-SADs is created to store column-SADs. LR, RR_i, and col-SADs are all implemented by shifting registers. For every N cycles, the right pixel memory RR_i performs one-pixel left column-shifting. And the col-SADs block also performs one-pixel left column-shifting. Thereafter, N vertical pixels are read in from each image. According to different disparities, the col-SAD results are stored into the last column of col-SADs block [82]. Finally, each row of col-SADs memory block is summarized and the smallest SAD corresponding disparity is obtained.

The packaged IP of depth calculation module is shown in Figure 5.17. The input data of this module are pixels from two images and the output is the disparity data for each image point. In the design, we discover that too much BRAM are required to buffer two N lines of pixels (N=nine in this IP). Therefore, in this module, the input image is rescaled to 160×120. Which is to say, read in four pixels, but only utilize one of them. Although the resolution of generated disparity image is 160×120, for the object that not too far away, the depth of the object calculated is still acceptable.

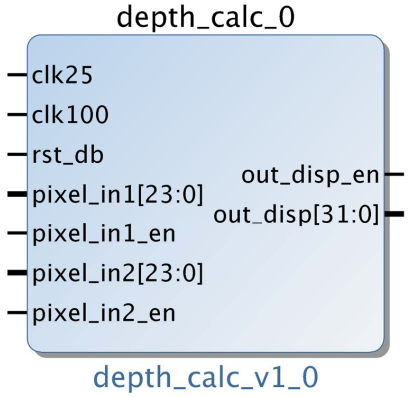


Figure 5.17: The packaged IP of depth calculation

Site Type	Resource Used	Available
LUT	5301	53200
FF	11788	106400
BRAM	14	140
DSP	0	220

Table 5.5: The resource utilization report of depth calculation IP on Zedboard

The resource utilization of depth calculation IP is listed in Table 5.5. Since many shifting registers are employed in this IP, many FFs and LUTs are used. Moreover, the resolution of the valid data in process block is 160×120, therefore, compared with rectification module, much less BRAMs are employed. Besides, it is worth noting that the output of this module is the disparity image. The process of depth calculation by applying Equation (5.5) is accomplished after the objects are detected.

5.5 Image Scaler

The task of image scaler is to scale the image data from one resolution to other resolutions, which can be used as the data input for feature calculation module. In the camera module, the OV7670 camera is configured to generate the output with 640×480 resolution. Therefore, the data input of image scalers in this section are all with 640×480 resolution. According to the fast feature calculation approach that shown in Figure 4.3, two image scalers are required: 320×240 scaler and 160×120 scaler, which is $2\times$ downsampled and $4\times$ downsampled image of the original image. Besides, the fast feature calculation process in practical can be re-drawn as Figure 5.18. In this section, two image scaler IP cores are created and explained.

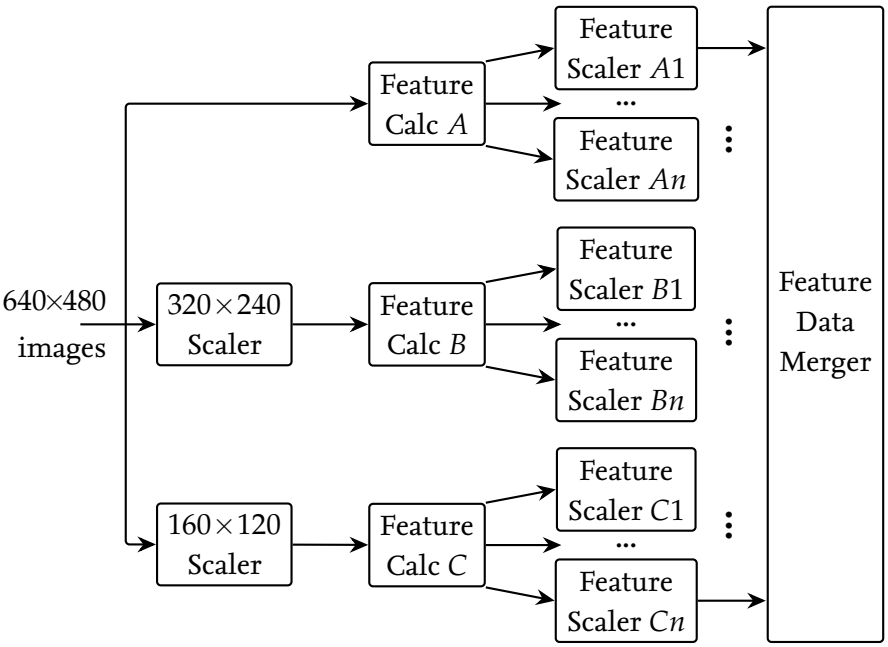


Figure 5.18: The structure of feature extraction module

5.5.1 320×240 Image Scaler

Because the resulting image of 320×240 image scaler is horizontally and vertically $2\times$ downsampled of the input image, minimum two lines of pixels are required to be buffered. In the design, to achieve reading from and writing to the buffer simultaneously, the size of BRAM buffer is configured to four lines of pixels: two lines for reading and two lines for writing. The input and output ports of 320×240 image scaler are shown in Figure 5.19.

The input clock frequency is 25MHz, and the input data is 24-bit pixel data. Thus,

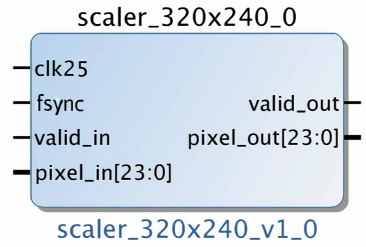


Figure 5.19: The packaged IP of 320 × 240 image scaler

each clock cycle, one 24-bit pixel is fed into the scaler. The output of this module is also 24-bit wide, but due to 2× horizontally and vertically downsampling, the output will be produced every four clock cycles. In figure 5.19, "fsync" is the frame synchronization signal, which indicates when current frame ends and next frame starts. This signal is similar to the "cam_vsync" port of camera and stereocam module.

Moreover, in this module, because the input pixel is streamed in every clock cycle, to accomplish data processing in time, a pipeline architecture is designed. The structure of the pipeline is illustrated in Figure 5.20. It is evident that this pipeline structure consists of five stages: writing input pixel to BRAM, determining when the process starts by buffered data validation, reading required data, summarizing the read data, and generating output signals. Worth to note that in stage 2, the quantity of buffered data will be checked, and based on this information, start/end processing signal will be generated. Thereafter, the whole pipeline starts until all pixels in the image is processed. Besides, because the resulting data is outputted every four cycles, each stage may finish its task in four cycles. For instance, in stage 4, pixels in the 2 × 2 window are requested to summarized in four cycles. Therefore, this pipeline architecture contains five stages where each stage may contain a maximum 4-states state machine.

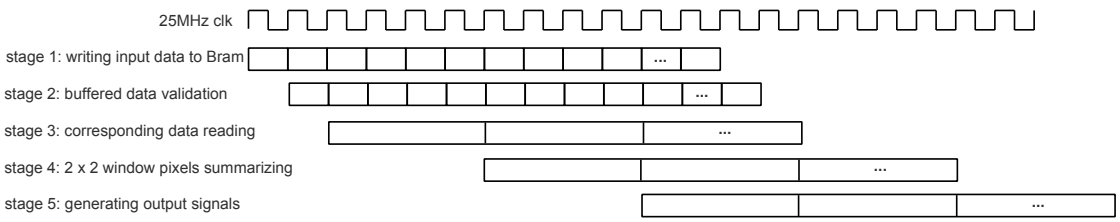


Figure 5.20: The pipeline structure of 320 × 240 scaler

5.5.2 160×120 Image Scaler

160 × 120 image scaler is similar to 320 × 240 image scaler, except the buffer size and computing process. Because each pixel in a 160 × 120 image is calculated from a 4 × 4 window, to guarantee simultaneous reading from and writing to BRAM, the buffer size is set to 8 × 640 where each pixel is 24-bit width. In stage 2 and 3, the pixels in 4 × 4 window will be accessed and summarized in 16 clock cycles. Hence, output data is produced every 16 clock cycles. Since the packaged IP and pipeline structure are quite similar to Figure 5.19 and 5.20, the figures for 160 × 120 image scaler are omitted.

The resource utilization report of 320 × 240 image scaler and 160 × 120 image scaler is listed in Table 5.6. From this table, we can conclude that similar quantity of LUTs and FFs are used by two image scalers. Because the buffer size of 160 × 120 image scaler is doubled, comparing to 320 × 240 scaler, the BRAMs used in 160 × 120 image scaler is also doubled.

Site Type	Resource Used		Available
	320 × 240 Scaler	160 × 120 Scaler	
LUT	203	213	53200
FF	295	307	106400
BRAM	2	4	140
DSP	0	0	220

Table 5.6: The resource utilization report of image scaler IP cores on Zedboard

5.6 Feature Calculation

The Feature calculation submodule, which is the real computing block according to the fast feature pyramid model, extracts ten channels' feature data from one image. As illustrated in Figure 5.18, feature calc A, B, and C are three feature calculation blocks which compute feature data from 640 × 480, 320 × 240, and 160 × 120 images, respectively. The inner structure of feature calculation module is shown in Figure 5.21.

For ease of description, in this part, we assume that the resolution of image input for this IP is 640 × 480. Firstly, 640 × 480 image is converted from RGB to CIELUV color space [21] by RGB2LUV block. Then, image filtering is performed for L, U, and V channel data by a triangle filter, respectively. Thereafter, magnitude and orientations of gradient are calculated for filtered L, U, and V channel data,

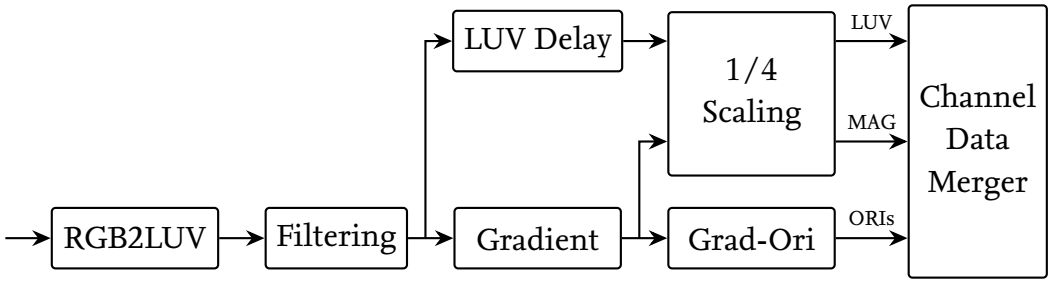


Figure 5.21: The inner structure of feature calculation submodule

respectively. The processing results from block Gradient are then transferred to block Grad-Ori, which is used to categorize gradient orientations to 6 groups by 4×4 window-based histogram. These 6 groups of data are actually the other 6 channel features used for detection.

Because the dimension of the resulting 6 grouped orientations from Grad-Ori is 160×120, the dimension of L, U, V and gradient-magnitude channels is 640×480, 1/4 scaling block is employed. In the end, features of 10 channels are combined and merged in channel data merger block.

5.6.1 RGB2LUV

Since the camera captured image is RGB formatted, the colorspace conversion from RGB to LUV CIE is firstly performed. In Section 2.2.1, the LUV feature and the RGB to LUV conversion process are introduced in detail. However, to implement the RGB2LUV conversion on FPGA platform, some problems should be considered and solved before applying the mathematic formulas introduced in Section 2.2.1. Firstly, conversion matrix in Equation (2.1) should be converted to integer by left shifting since no floating point operation is supported by Verilog/VHDL. Moreover, cube root operation is required in luminance L_i calculation and division operation is required in Equation (2.3). Divider generator IP core [83] provided by Xilinx, can be employed to do the division operation with the cost of thousands of FFs and LUTs consumption. Beside, no cube root IP core is available.

To generate efficient pipeline architecture, in this submodule, lookup table is used for both cube root and division operation. The reason lookup table can be used as the replacement for them is that the data range of L_i, U_i and V_i are limited. For luminance, the data range of L_i is between 0 to 100. And the range of U_i and V_n is [-88, 182] and [-134, 105], respectively. If we denote the lookup table used for cube root and division as LUT_c and LUT_d, respectively. Equations (2.2) and (2.3) can be

rewrite to:

$$L_i = LUT_c(Y_i) \tag{5.7}$$

and

$$\begin{aligned} U_i &= 13L_i(LUT_d(X_i, Y_i, Z_i) - 0.197833) \\ V_i &= 13L_i(LUT_d(X_i, Y_i, Z_i) - 0.468331), \end{aligned} \tag{5.8}$$

respectively. In the design, to simplify the computing complexity of future modules that using LUV data, L_i, U_i and V_i are normalized to avoid the negative values. In the end, this submodule is also packaged as an IP in my toolbox, as shown in Figure 5.22.

IP Packaging and Resource Utilization Analysis

The 25MHz clock is used as the input clock of pixel data, and 100MHz is used as clock of all computing blocks. The output from this IP is the L, U, and V channel data, which in altogether are 30-bit width. Because the ranges of L, U, and V after normalization and left-shifting are [0, 100], [0, 270] and [0, 239], respectively, they can be represented by 7-, 9- and 8-bit data. Besides, to convert the floating point operation to integer computing, left shifting is also applied for other computations. For instance, 16 bits left-shifting is used for RGB to CIEXYZ color space conversion according to Equation (2.1).

Moreover, approximating cube root and division operation also enlarged the real data by around 10 bits. In order to maintain the precision in channel filtering and gradient computing submodules, two extra bits are kept for each channel data. Therefore, the output "pixel_out" signal contains 9-bit L, 11-bit U, and 10-bit V.

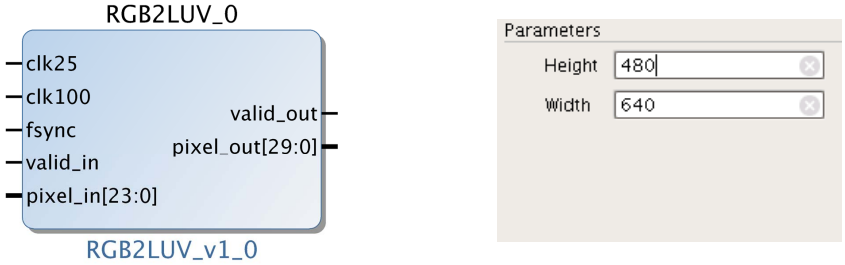


Figure 5.22: The packaged IP and its configurable parameters of RGB2LUV submodule

Because feature calculation process should be repeated under different image scales, the width and height of input image of this IP are customizable. By configuring these two parameters, RGB2LUV IP could achieve color space conversion for any resolution images. The resource utilization of this submodule is listed in

Site Type	Resource Used	Available
LUT	802	53200
FF	888	106400
BRAM	3.5	140
DSP	4	220

Table 5.7: The resource utilization report of RGB2LUV IP on Zedboard

Table 5.7. 3.5 BRAMs are utilized for composing cube root and division lookup table. And DSPs are used during the U_i and V_i computing process according to Equation (5.8).

5.6.2 Channel Filtering

After the L, U, and V channel features are obtained, filtering is performed for each channel. The intention of this block is to reduce data noise which exist or is produced via color space conversion. In ACF paper [34], a 11×11 triangle filter Δ, as shown in Equation (5.9), is employed during this process.

$$\Delta = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} \tag{5.9}$$

To realize 11×11 window based filtering on FPGA, at least 11 lines of 640×30-bit data should be buffered. Considering the limited BRAM resources on FPGA, only a 3×3 triangle filter Δ' is used in this submodule.

$$\Delta' = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{5.10}$$

Moreover, the triangle filter Δ' for the upper-left, upper-right, lower-left, and

lower-right corner of any channel image can be altered to:

$$\Delta' = \begin{bmatrix} 9 & 3 \\ 3 & 1 \end{bmatrix} or \begin{bmatrix} 3 & 9 \\ 1 & 3 \end{bmatrix} or \begin{bmatrix} 3 & 1 \\ 9 & 3 \end{bmatrix} or \begin{bmatrix} 1 & 3 \\ 3 & 9 \end{bmatrix}. \tag{5.11}$$

In addition, for the left, right, upper, and lower boundary of the channel image, Δ' satisfies

$$\Delta' = \begin{bmatrix} 3 & 1 \\ 6 & 2 \\ 3 & 1 \end{bmatrix} or \begin{bmatrix} 1 & 3 \\ 2 & 6 \\ 1 & 3 \end{bmatrix} or \begin{bmatrix} 3 & 6 & 3 \\ 1 & 2 & 1 \end{bmatrix} or \begin{bmatrix} 1 & 2 & 1 \\ 3 & 6 & 3 \end{bmatrix}. \tag{5.12}$$

Because the dimension of input data and output data is same, the input data frequency is 25MHz and processing clock is 100MHz, each data must be processed within four clock cycles. Nevertheless, for the calculation of each point, 9 pixels inside the 3x3 window are required, which is impossible to read out in four cycles. To fix this problem, the input data are directly processed by a [1 2 1] filter, and then the filtered results are stored to BRAM. In this way, for each point, only three data are required to be fetched from BRAM in four clock cycles. Besides, row-by-row calculation and keeping intermediate results methodology is insensitive to the filter size, therefore, channel filtering with a 11×11 triangle filter can also be implemented via this approach.

IP Packaging and Resource Utilization Analysis

The resource utilization of channel filtering IP is shown in Table 5.8. Although only 3×3 triangle filter-based channel filtering is used in my application design, both 11×11 and 3×3 triangle filter-based channel filtering IP are created. From Table 5.8, it is evident that the BRAM utilization of channel filtering IP with 11×11 triangle filter is tripled, compared to the 3×3 version.

Site Type	Resource Used		Available
	3×3 Triangle Filter	11×11 Triangle Filter	
LUT	423	996	53200
FF	500	810	106400
BRAM	10	12	140
DSP	0	0	220

Table 5.8: The resource utilization report of channel filtering IPs on Zedboard

The packaged IP of this submodule is shown in Figure 5.23. Three parameters are configurable: image height, image width, and interval cycles between input data.

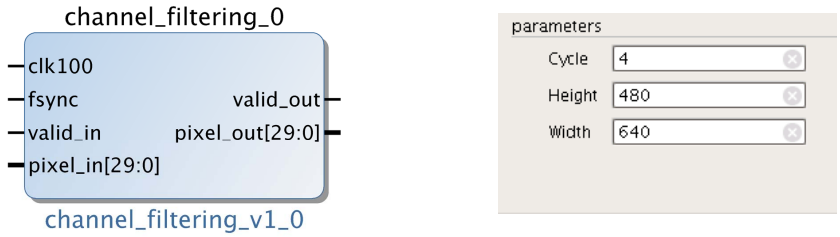


Figure 5.23: The packaged IP of channel filtering submodule

The parameters image height and width make this IP suitable for the data input with any resolutions. In order to guarantee that the input data can be processed in real-time, parameter "Cycle", which indicates the interval cycles of input data, is provided. Note that the minimum interval cycles of input data is 4 cycles of 100MHz clock when the input data is generated from RGB2LUV IP with 25MHz clock.

5.6.3 Gradient Computing

The job of gradient computing submodule is to compute gradient magnitude and orientation for filtered L, U, and V channel data. Assuming for any point $P(x,y)$, its corresponding feature value after filtering are $L(x,y)$, $U(x,y)$, and $V(x,y)$, the gradient magnitude $M_L(x,y)$ of point $P(x,y)$ in L channel could be calculated by:

$$\begin{aligned} M_{Ldx}(x,y) &= L(x+1,y) - L(x-1,y) \\ M_{Ldy}(x,y) &= L(x,y+1) - L(x,y-1) \\ M_L(x,y) &= \sqrt{M_{Ldx}(x,y)^2 + M_{Ldy}(x,y)^2} \end{aligned} \tag{5.13}$$

where $L(x+1,y)$, $L(x-1,y)$, $L(x,y+1)$, $L(x,y-1)$ are the L channel values of Point P's surrounding points. $M_{Ldx}(x,y)$ and $M_{Ldx}(x,y)$ are the gradient in x-axis and y-axis direction, respectively. With the same equation, magnitude in U channel $M_U(x,y)$ and in V channel $M_V(x,y)$ could also be obtained.

Thereafter, the final gradient magnitude $M(x,y)$ of point $P(x,y)$ can be computed by:

$$M(x,y) = max(M_L(x,y), M_U(x,y), M_V(x,y)), \tag{5.14}$$

which is the process of getting maximum value between three channels. In the end, the orientation $O(x,y)$ can also be calculated by:

$$O(x,y) = arctan(\frac{M_{dy}(x,y)}{M_{dx}(x,y)}), \tag{5.15}$$

where $M_{dy}(x,y)$ and $M_{dx}(x,y)$ are the final gradient magnitude values in y-axis and x-axis direction.

Equations (5.13) to (5.15) show the calculation process in theory or in Matlab software. However, for the hardware implementation, some questions should be figured out before the design. In Equation (5.13), square root operation is required, and this equation will be executed three times for each pixel. Besides, after calculation in Matlab, the experimental results show that the range of $M_L(x,y)$, $M_U(x,y)$ and $M_V(x,y)$ is very wide. Therefore, it is impossible to use a lookup table to replace square root operation. Hence, CORDIC IP core [84] provided by Xilinx Vivado is employed to achieve square root calculation.

Moreover, in orientation calculation, arctan operation should be executed. Although an IP core for arctan is also provided by Xilinx, the latency and resource usage is relatively high. Because arctan can be transformed to arccosine whose data range is between -1 and 1, a lookup table is used in this block to replace the arctan operation. If we denote the lookup table as LUT, the orientation calculation equation (5.15) is changed to:

$$\begin{aligned} O(x,y) =& LUT(\frac{M_{dy}(x,y)}{M_{dx}(x,y)}) \\ \Rightarrow& LUT(\frac{M_{dx}(x,y)}{M(x,y)}) \\ \Rightarrow& LUT_d(M_{dx}(x,y) \cdot LUT_{arccos}(M(x,y))), \end{aligned} \tag{5.16}$$

where LUT_{arccos} is the arccosine operation lookup table and LUT_d is the division lookup table, which is used to further simplify the orientation computing process. Until now, the gradient magnitude and orientation are calculated for each point. Because the results of gradient magnitude are normally very small, normalization process is executed for gradient magnitude. Before normalization, in order to remove the errors introduced by approximation of lookup table and square root IP, magnitude results are passed by a 3×3 triangle filter first. Then, the normalization of magnitude is processed by:

$$M_i(x,y) = \frac{M(x,y)}{c + M_{tri}(x,y)} \ , \ c = 0.005 \tag{5.17}$$

where $M_{tri}(x,y)$ is the magnitude result after triangle filtering and $M_i(x,y)$ is the normalized magnitude at point $P(x,y)$. During the normalization, a calibration parameter c, which normally equals to 0.005 is employed.

Due to the complexity of this IP, in the design, three blocks are created to achieve gradient magnitude and orientation computing, as shown in Figure 5.24. “grad_m”

block calculates the squared gradient magnitude $M_L^2(x,y)$, $M_U^2(x,y)$, and $M_V^2(x,y)$ for L, U, and V channel, respectively, according to Equation (5.13). Then the maximum value of $M_L^2(x,y)$, $M_U^2(x,y)$, and $M_V^2(x,y)$ are selected for each point.

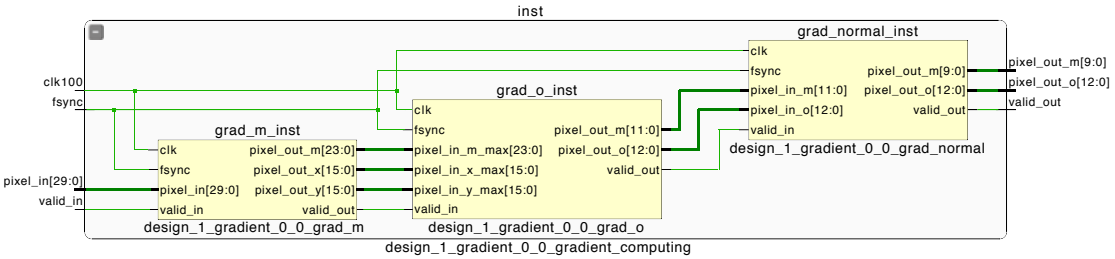


Figure 5.24: The inner structure of gradient computing IP

With squared gradient magnitude information, square root operation is performed in "grad_o" block by CORDIC IP [84]. After that, orientation is computed through two times searching in arccosine lookup table LUT_{arccos} and division lookup table LUT_d. In the end, "grad_normal" block performs normalization of gradient magnitude, which contains triangle filtering for the results obtained from "grad_o" block and division according to Equation (5.17).

The implementation approach of triangle filtering used in this part is similar to the channel filtering IP created in Section 5.6.2, except that the width of input data is different and only one channel is necessary to be filtered in this block. Through data analysis, the results show that division operation in normalization block cannot be replaced. Therefore, the IP core Divider Generator [83] is used. Besides, because orientation data does not need to be normalized and 20 clock cycles latency of magnitude data by division operation, a shifting register based delay block is added for orientation data output.

Note that in Figure 5.24, the bit-width of squared gradient magnitude "pixel_out_m" is 24-bit and the data width of gradient in x-, y-axis direction is 16-bit. These widths are set up by data analysis from Matlab results. Nevertheless, because the precision loss is quite small in "grad_normal" block, to save hardware resources used by triangle filtering, 12-bit and 13-bit are kept for gradient magnitude and orientation after "grad_o" block.

Latency

For hardware implementation on FPGA, the latency of each IP/module can be calculated. However, because this information is required for further design of other submodules, the latency of gradient computing IP, Grad-Ori IP and 1/4 scaling IP is explained in detail in this thesis. The latency of this submodule consists of three

parts: the latency of "grad_m" block, the latency of "grad_o" block, and the latency of "grad_normal" block.

- The latency of "grad_m" block depends on the dimension of input channel data. More precisely, it is related to the data width of input image, which is the configurable parameter *"Width"* of gradient computing IP. $(Width + 2)$ pixels are required to be buffered before the computing starts in "grad_m" block. Assuming the input clock is 100MHz and the input data interval is 4 clock cycle. Then the latency of input data buffering is $(Width + 2) \times 4$. Besides, a five-stage pipeline architecture is the processing block of "grad_m" whose latency is 14 clock cycles. Therefore, $(Width + 2) \times 4 + 14$ is the latency of 'grad_m" block.

- The "grad_o" block contains one CORDIC IP for square root calculation and other logics to complete gradient orientation computation. The latency of calculating square root is 13 clock cycles and the rest logics takes 12 clock cycles. Thus, the latency of "grad_o" block is 25 clock cycles.

- "grad_normal" block contains a triangle filtering and a division IP core. The latency of triangle filtering also depends on the parameter *Width*, which is equal to $(Width + 2) \times 4 + 14$. And 20 clock cycles' delay is produced by the divider IP. Therefore, the latency of this block is $(Width + 2) \times 4 + 14 + 20$.

After the analysis of latency for each block inside this submodule, the total latency of gradient computing IP can be calculated by summarizing the latency of three blocks, which is equal to $(2Width + 4) \times 4 + 73$. The latency calculation is under the assumption of 100MHz clock input and continuous data input with the interval of 4 clock cycles. In reality, because the resolution of input image is changeable, the interval of data input can be bigger than four. Therefore, a generalized latency $L_{gradient}$ of this IP would be:

$$L_{gradient} = (2Width + 4) \times C_{interval} + 73, \quad \text{with} \quad C_{interval} \geq 4 \tag{5.18}$$

where $C_{interval}$ is the clock interval of input data.

IP Packaging and Resource Utilization Analysis

The packaged IP of this submodule is shown in Figure 5.25. The configurable parameters of this IP is the same as channel filtering IP that described in Section 5.6.2.

The resource utilization of this IP and its inner blocks are listed in Table 5.9. Because the gradient calculation in y-axis direction required at least three lines of data and one extra line is used for new input data buffering. Therefore, a number

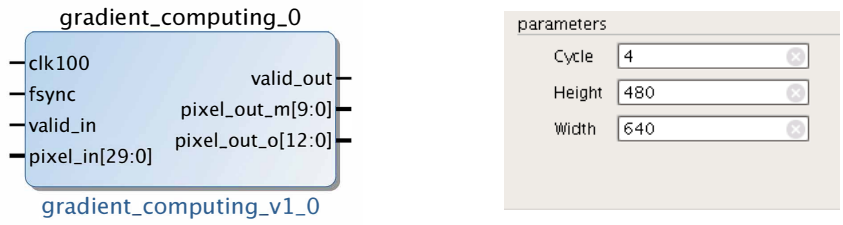


Figure 5.25: The packaged IP of gradient computing submodule

of 3.5 BRAMs is used in "grad_m" block. In "grad_o" block, two lookup tables are created with BRAM, thus, 3.5 BRAMs are consumed in this block. Besides, three BRAMs are also consumed in normalization block for triangle filtering. With respect to DSP usage, square operation in Equation (5.13) requires four DSPs. Calculating the input for arccosine lookup table consumes one DSP in "grad_o" block. And one more is employed during the normalization process.

Site Type	Resource Used				Available
	grad_m	grad_o	grad_normal	gradient IP	
LUT	702	313	540	1555	53200
FF	403	363	1227	1993	106400
BRAM	3.5	3.5	3	10	140
DSP	4	1	1	6	220

Table 5.9: The resource utilization report of gradient computing IP on Zedboard

5.6.4 Grad-Ori Computing

The gradient computing IP computes gradient magnitude and orientation for the L, U and V channel of an image. For the comparison of image features, it is reasonable to select gradient magnitude as one channel feature. However, for orientation, because slightly difference between two image points can result in obvious orientation difference, some post-processing is required to enhance the robustness of gradient orientation feature. According to ACF algorithm [34], the gradient orientation results are categorized into six groups/bins O_1, O_2, O_3, O_4, O_5, and O_6. Then the gradient orientation $O(x, y)$ of any point could be expressed by its two nearest bins.

In practical, gradient magnitude is used to realize this orientation quantization. Because for any point $P(x, y)$, the data range of its corresponding gradient orientation $O(x, y)$ is always between $0°$ and $180°$, the bin interval is $30°$. For instance, if O_1 is set to $0°$, the other bins are located at $30°$, $60°$, 90^o, 120^o, 150^o. If we denote

the gradient orientation and magnitude of one point as O' and M', its nearest bins are O_i and O_j, then

$$M_i' = M' \cdot \mid O_i - O' \mid, \quad M_j^2 = M' \cdot \mid O_j - O' \mid \tag{5.19}$$

where M_i' and M_j' are the results that M' is divided into two bins by orientation quantization.

With this categorizing strategy, gradient orientation information is much more robust for feature comparison. Nevertheless, the result of each point consists of six components, although four of them are zero and only two are valid. In order to reduce the computation complexity when using six components, a 4×4 window based histogram is processed. Then six accumulated bin results are produced by every 4×4 window. In other words, the quantity of the output data is 1/16 of the input data. For instance, if the dimension of input data is 640×480, the dimension of output data is only 160×120. Although the results of this IP are still six 10-bit accumulation components, the data output frequency is remarkably reduced.

Latency

Assuming the input clock is 100MHz, the interval of data input is 4 clock cycles, and the width of input image is denoted as $Width$. The latency of this IP contains two parts: the delay of data buffering and the latency of processing block, which can be calculated as follows.

- Because 4×4 window based histogram is processed in this IP, minimum three lines of pixels plus 14 should be buffered. Thus, the latency of this part is equal to $(3Width - 1 + 15) \times 4$ clock cycles.
- For the processing block, a five-stage pipeline structure is designed. The delay of this pipeline architecture is 16 clock cycles.

By summarizing the delay of these two parts, the total latency of Grad-Ori IP is obtained, which equals $(3Width + 14) \times 4 + 16$ clock cycles. Besides, a generalized latency $L_{gradient}$ of this IP would be:

$$L_{gradient} = (3Width + 14) \times C_{interval} + 16, \quad \text{with} \quad C_{interval} \geq 4, \tag{5.20}$$

where $C_{interval}$ is the clock interval of input data.

IP Packaging and Resource Utilization Analysis

The packaged IP and its configurable parameters are shown in Figure 5.26. The configurable parameters of this IP are the same as channel filtering IP that introduced in Section 5.6.2.

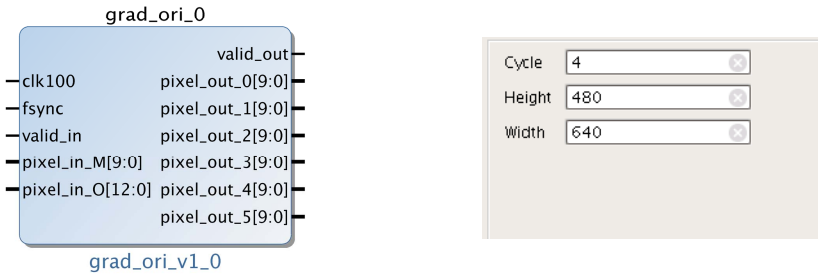


Figure 5.26: The packaged IP of grad_ori computing submodule

The clock rate of processing block is 100MHz and input data interval is 4 clock cycles. The input data are 10-bit gradient magnitude and 13-bit gradient orientation that obtained from gradient computing submodule. And six accumulated orientation results that generated by a 4×4 window based histogram are the data output. The resource utilization of grad_ori IP is shown in Table 5.10. To buffer the orientation data for the 4×4 window based histogram, a number of 7 BRAMs is used. And DSPs are consumed by the weighted magnitude multiplication according to Equation (5.19).

Site Type	Resource Used	Available
LUT	453	53200
FF	597	106400
BRAM	7	140
DSP	2	220

Table 5.10: The resource utilization report of Grad-Ori IP on Zedbaord

5.6.5 1/4 Scaling

For the input RGB image, after RGB2LUV and triangle filtering block, L, U, and V channel features are calculated. By Gradient Computing submodule, gradient magnitude channel feature is obtained. And with Grad-Ori block, six channels of gradient orientation are computed. Nevertheless, the dimension of these channels is not the same. For a 640×480 input image, the dimension of L, U, V and gradient magnitude channel is 640×480. However, the dimension of gradient-orientation channels that produced from Grad-Ori IP is 160×120.

To use these 10 channels' feature for object classification easily, a 1/4 scaling IP is designed to downsample the L, U, V, and gradient magnitude channel data from 640×480 to 160×120, as mentioned in Figure 5.21. During the downsampling process, bilinear interpolation based scaling approach is employed. Besides,

1/4 scaling, vertically and horizontally, is a special case for bilinear interpolation where each point in scaled image is the mean of a 4×4 window in the 640×480 image.

Latency

The latency of this IP is similar to Grad-Ori IP. Namely, two parts are included: data buffering and data processing. For input data buffering, because data in a 4×4 window are required, if we denote the clock interval of the input data as $C_{interval}$, $(3Width - 1 + 15) \times C_{interval}$ clock cycles should be waited before the processing starts. And the latency of data processing part is 14 clock cycles. Therefore, the overall latency of 1/4 scaling submodule is $(3Width - 1 + 15) \times C_{interval} + 14$ clock cycles, with $C_{interval} \geq 4$.

IP Packaging and Resource Utilization Analysis

The interface of 1/4 scaling IP and its configurable parameters are shown in Figure 5.27. The configurable parameters of this IP are the same as channel filtering IP that introduced in Section 5.6.2. 100MHz is used as the clock of processing

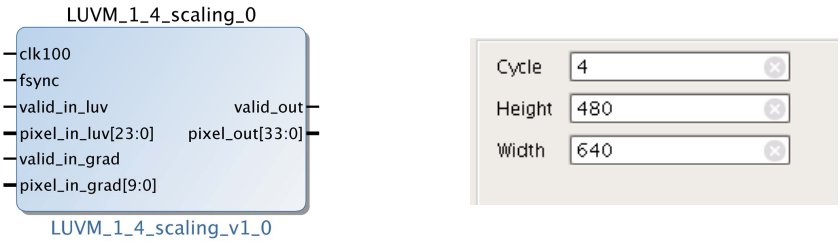


Figure 5.27: The packaged IP of 1/4 scaling submodule

block and one fsync bit is used for frame synchronization. 24-bit LUV and 10-bit gradient magnitude are the IP input with a minimum input interval of four clock cycles. The output of this submodule is a 34-bit scaled data, which combines 24-bit LUV and 10-bit gradient magnitude. The resource utilization of this submodule is shown in Table 5.11. Because 4×4 window based averaging is processed for the input data, 8 lines of data are buffered where each line is 640×33-bit. Therefore, as Table 5.11 shows, five BRAMs are used in this submodule.

5.6.6 LUV Delay

In Section 5.6.5, L, U, V and gradient magnitude are downsampled. However, the gradient magnitude is computed from L, U, and V channel data and the latency of gradient computing IP is $(2Width + 4) \times 4 + 73$. In order to guarantee the L, U, V and gradient magnitude arrives to 1/4 scaling IP as the same clock cycle, a delay

Site Type	Resource Used	Available
LUT	246	53200
FF	423	106400
BRAM	5	140
DSP	0	220

Table 5.11: The resource utilization report of Grad-Ori IP on Zedboard

submodule for LUV channel data is required, as shown in Figure 5.21. The clock cycles that should be delayed by this IP are the same as the latency of gradient computing IP, namely, $(2Width + 4) \times 4 + 73$.

In the design, it is impossible to use shifting registers to delay two lines of 640 feature data. Therefore, BRAM based delay buffer is the option. However, if we divide the latency into two parts: $(2Width + 4) \times 4$ and 73, we notice that 73 is the fixed latency of data processing blocks in Gradient IP. Meanwhile, $(2Width + 4) \times 4$ is the latency required for data buffering. Thus, in reality, for simplifying the BRAM configuration process, two delay blocks are created, as illustrated in Figure 5.28.

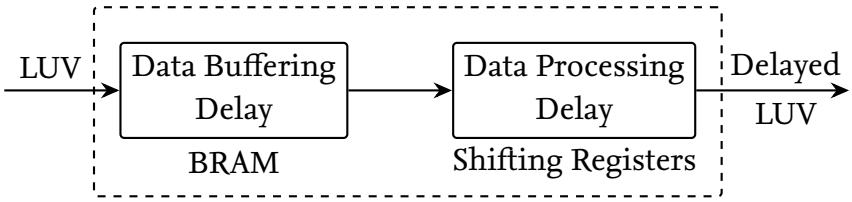


Figure 5.28: The structure of LUV delay submodule

Data buffering delay block employs BRAM to provide the latency of line buffering part and data processing delay block uses shifting registers to provide the fixed processing latency. Although the parameter *Width* is changeable, for different scaled image input, only calculating the latency of line buffering is much easier for BRAM configuration and utilization.

The overall delay process consists of a three stage pipeline structure. The first stage stores input data to BRAM and determines when the second stage starts. The second stage fetches data from BRAM and the final stage delays a few cycles by shifting registers. Due to the parameter *Width* is configurable, the interval of data input changes. In order to always generate correct start signal for the second stage, a new parameter "Delay input number" is introduced, as shown in Figure 5.29.

For instance, the processing block clock is 100MHz and the input image is 160 × 120, then the clock interval of input data is 64 clock cycles. Assuming the "Delay Input Number" is 1, then after 64 × 1 cycles, stage 2 starts. Meanwhile, when the input image is 320 × 240, the corresponding clock interval of input data would be 16 clock cycles. To guarantee the second stage also starts after 64 cycles, four input data should be received firstly. Therefore, "Delay Input Number" is set to 4. And for 640 × 480 image, this parameter is equal to 64.

IP Packaging and Resource Utilization Analysis

Figure 5.29 shows the packaged IP of LUV Delay submodule and its configurable parameters. The input data and output data are both 24-bit LUV feature data. 100MHz clock is used as the processing clock and one 'fsync' signal is used for frame synchronization. The resource utilization report of LUV delay submodule

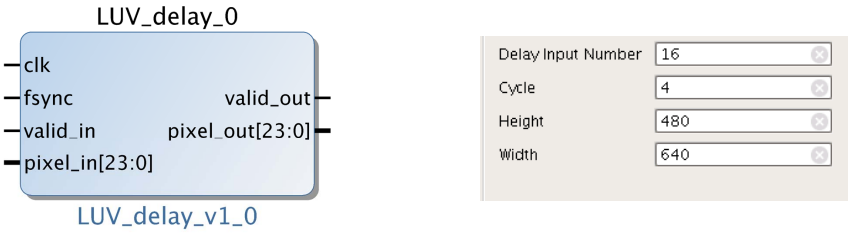


Figure 5.29: The packaged IP of LUV delay submodule

is listed in Tab 5.12. In the design, to guarantee the BRAM size is enough for data input with different dimensions, the size of BRAM buffer is set to three lines of 640 data that each is 24-bit width. Thus, a reasonable number of BRAM is used by this IP.

Site Type	Resource Used	Available
LUT	138	53200
FF	271	106400
BRAM	1.5	140
DSP	0	220

Table 5.12: The resource utilization report of LUV Delay IP on Zedboard

5.6.7 Channel Data Merger

After feature calculation and feature scaling, features of 10 channels, which include L, U, V, gradient magnitude and 6 gradient orientations are all obtained. In this

section, data merger submodule is created, which combines 10 channels' feature data together.

From 1/4 feature scaling IP, we know that LUV and magnitude channel feature are outputted concurrent. However, the latency of 1/4 scaling IP is $(3Width - 1 + 15) \times C_{interval} + 14$ and the latency of Grad-Ori IP is $(3Width - 1 + 15) \times C_{interval} + 16$. Two cycles' difference exists between them, which is solved by shifiting registers in channel data merger IP.

IP Packaging and Resource Utilization Analysis

The packaged IP of channel data merger is shown in Figure 5.30. "luvm_pixel_in" is the data input port for L, U, V and gradient magnitude. "ori_pixel_in1" to "ori_pixel_in6" are the six gradient orientation input ports. And a combined 94-bit data is outputted from "pixel_out" port. The resource utilization of this IP is listed in Table 5.13. Since this IP is quite simple, only combining 10 data to one, and only few resources are occupied.

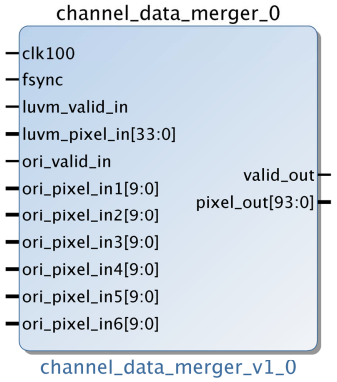


Figure 5.30: The packaged IP of channel data merger submodule

Site Type	Resource Used	Available
LUT	2	53200
FF	191	106400
BRAM	0	140
DSP	0	220

Table 5.13: The resource utilization report of channel data merger IP on Zedboard

5.6.8 IP Packaging and Resource Utilization Analysis

From Section 5.6.1 to 5.6.7, all IP cores that inside feature calculation module are introduced. With these IPs, the feature calculation IP is created according to Figure 5.21. The connections between these IP cores are shown in Figure 5.31. From this figure, it is evident that four extra units, 3 slices and one concat, are used in this design.

In the original algorithm, all intermediate results are floating point, especially in RGB2LUV and gradient computing IP. In the hardware design, left/right-shifting is used to convert the floating point operation to integer operation. Besides, lookup

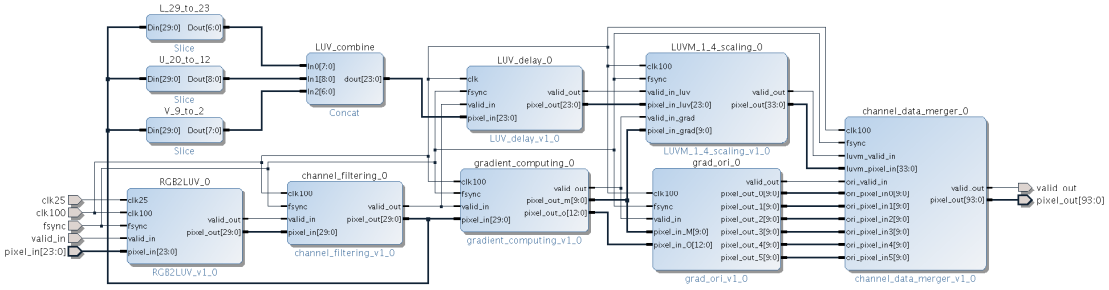


Figure 5.31: The connections that inside feature calculation IP

table is also employed to replace some division, arctan operation. Therefore, the L, U, and V feature results from channel filtering IP are kept with 2-bit extra to maintain higher precision for gradient computing block. Nevertheless, LUV Delay module does not require these two extra bits. To save the buffer size used in LUV delay IP, these 2-bit for L, U, and V channel are removed by employing slices and concat unit. In this work, Figure 5.31 illustrates block design is also packaged into one IP, namely, feature calculation IP. The packaged IP and its configurable paramters are shown in Figure 5.32.

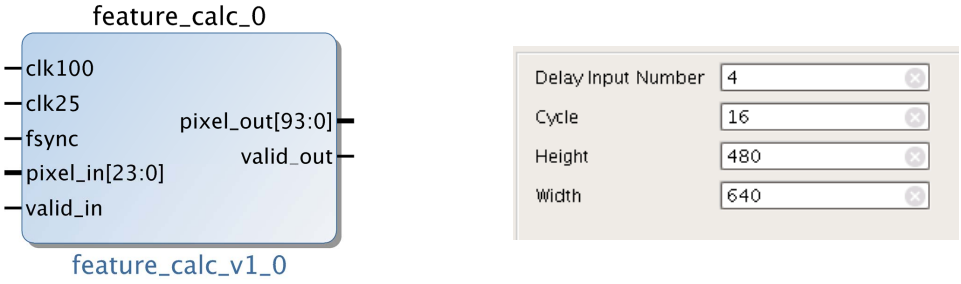


Figure 5.32: The packaged IP of feature calculation module

Two input clocks are used in this IP: 25MHz and 100MHz. 25Mhz clock is used only for RGB2LUV IP, which serves as the clock of data input. Meanwhile, the processing block in RGB2LUV and all other IP cores inside feature calculation module employs 100MHz as the main clock. Moreover, one bit “fsync” signal is used to indicate frame synchronization. The data input is 24-bit RGB pixel, and 94-bit feature data is outputted after feature calculation. The configurable parameters of this IP are also listed in Figure 5.32, which is the same as the LUV delay IP that introduced in Section 5.6.6. Therefore, the detailed explaination of parameter setup is omitted here.

The overall resource utilization of feature calculation module is listed in Table

5.14.

Type	Resource Used								Available
	RGB2LUV	Filtering	Grad	Grad-Ori	LUV Delay	1/4 Scaling	Merger	Sum	
LUT	802	423	1712	453	138	246	2	3776	53200
FF	888	500	2052	597	271	423	191	4923	106400
BRAM	3.5	4	10	7	1.5	5	0	31	140
DSP	4	1	6	2	0	0	0	12	220

Table 5.14: The resource utilization report of feature calculation IP on Zedboard

Due to the computation complexity and buffer requirement for each IP, 31 BRAMs are used, which is a lot since a few feature calculation IPs are required for an object detection application. On the other hand, the LUTs, Filp-Flops, and DSPs utilized by this IP are acceptable. Besides, by customizing this IP with different configurable parameters, the resources consumed by this IP may change slightly.

5.7 Feature Scaler

Feature calculation module calculates the ten channels' feature data from the input image, which is the real computing process according to the fast feature pyramid model. Therefore, many resources, especially BRAMs are used during the implementation of feature calculation IP. Meanwhile, the approximate computing process, which calculates the feature data by applying bilinear interpolation on the real computing results, is implemented in this section.

Although 1/4 Scaling IP and Image Scaler IP are also based on bilinear interpolation, since they are all the special cases that can be easily down-sampled by $2\times$, $4\times$, the implementation of bilinear interpolation in feature scaler module is quite different. According to the fast feature pyramid theory, the parameters of feature approximation under each image scale can be obtained in MATLAB. These parameters are set up as the configurable parameters for feature scaler IP. These parameters and their functions are listed in Table 5.15.

In the algorithm, the scaling ratios $scale\_x$, $scale\_y$, $sinV\_x$, and $sinV\_y$ are all decimal. In order to implement feature approximate computing, all these parameters are left-shifted 16 bits. Moreover, the dimension of scaled images varies a lot. For instance, assuming the resolution of the input image is 320×120, the dimension of calculated feature image is 80×60. And for a 640×480 input image, the resulting feature image is 160×120.

During the bilinear interpolation, because always two nearby pixels are required

Parameter	Function
Cycle IN	The time interval of input data (clock cycles)
Cycle OUT	The time interval of output data (clock cycles)
H_original	The height of input image data (from real computing)
W_original	The width of input image data (from real computing)
H_target	The height of output image data (after approximate computing)
W_target	The width of output image data (after approximate computing)
lambda	Obtained from Matlab, used for data approximation
scale_x	The scale ratio of W_target/W_original
scale_y	The scale ratio of H_target/H_original
sinV_x	The inverse of scale_x
sinV_y	The inverse of scale_y

Table 5.15: The configurable parameters of feature scaler IP

for each point calculation, minimum two lines of pixels should be buffered. In the design, to guarantee the buffering reading and writing simultaneously, the buffer size of each data scaler is set up to four lines of feature data, where each data is 94 bit. Therefore, for a 80×60 feature image, the amount of buffered data is $4 \times 80 \times 94$-bit while $4 \times 160 \times 94$-bit for a 160×120 feature image. To save BRAM resources on Zedboard, Xilinx Parameterized Macros (XPM) based BRAM configuration approach, which was firstly introduced in Vivado 2016.1 is employed. With this approach, the BRAM size can be configured by the parameter "W_original".

IP Packaging and Resource Utilization Analysis

The packaged IP of this module and its corresponding configurable parameters are shown in Figure 5.33. 100MHz is used as the input clock and one bit "fsync" is used as the frame synchronization signal. The 94-bit feature data from feature calculation module serves as the data input and the resulting scaled feature data is also 94 bit.

Due to the differences of configuration settings, the resource used by this IP varies. Table 1 listed the resource utilization report for some feature scalers. Because 135×101 feature scaler is computed using 160×120 feature calculation results, comparing to the other two scalers, 3 BRAMs are used. Moreover, due to the difference of "Cycle_IN" and "Cycle_OUT" parameters, the utilization of LUTs and FFs are also slight different. In this IP, many multiplication operations are required to achieve bilinear interpolation, therefore, 15 DSPs are consumed for each feature scaler.

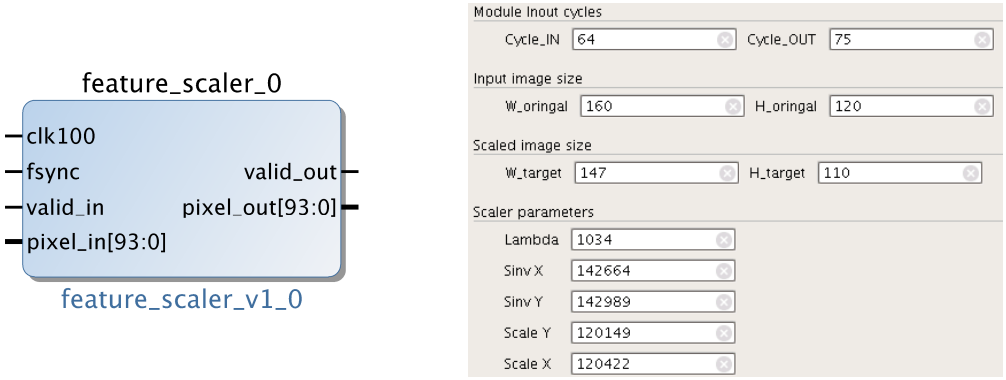


Figure 5.33: The packaged IP of feature scaler module

Site Type	Resource Used			Available
	135×101 feature scaler	95×71 feature scaler	67×50 feature scaler	
LUT	622	643	608	53200
FF	857	880	849	106400
BRAM	3	1.5	1.5	140
DSP	15	15	15	220

Table 5.16: Resource utilization report of different feature scaler IPs on Zedboard

5.8 Feature Data Merger

Feature data merger is the module which combines all feature data computed from real computing IPs and approximate computing IPs together and then transfer these data to the memory. Because the dimensions of each feature calc module and feature scaler module are all different, therefore, the feature data that feature data merger module received are out-of-order. In order to differentiate which feature data is received from which IP, a scale index is added to each channel.

The second task of channel data merger is to convert 94-bit data to a group of 32-bit data and transfer them to DDR3 memory via VDMA. The bit-width of each feature data is 94 bit. For a 640×480 input image, its corresponding feature image is 160×120. It is impossible to store all these data in BRAM, especially the feature data will be extracted under different image scales. Therefore, transferring these data to DDR3 memory via VDMA is required.

In the design, since 94-bit feature data contains 10 channels' information, each channel data is reformed to 16 bit. Then five 32-bit data are generated from each 94-bit data. Besides, one extra 32-bit data is created to represent the scale index.

In the end, six 32-bit data are required to be transferred via VDMA for each 94-bit feature data. Moreover, converting 94-bit feature data to separated 16-bit format also simplifies the feature access process in detection module.

To generalize this IP to support different numbers of inputting scalers, this IP is designed to accept feature data from maximum ten scalers. The packaged IP of feature data merger module is shown in Figure 5.34.

IP Packaging and Resource Utilization Analysis

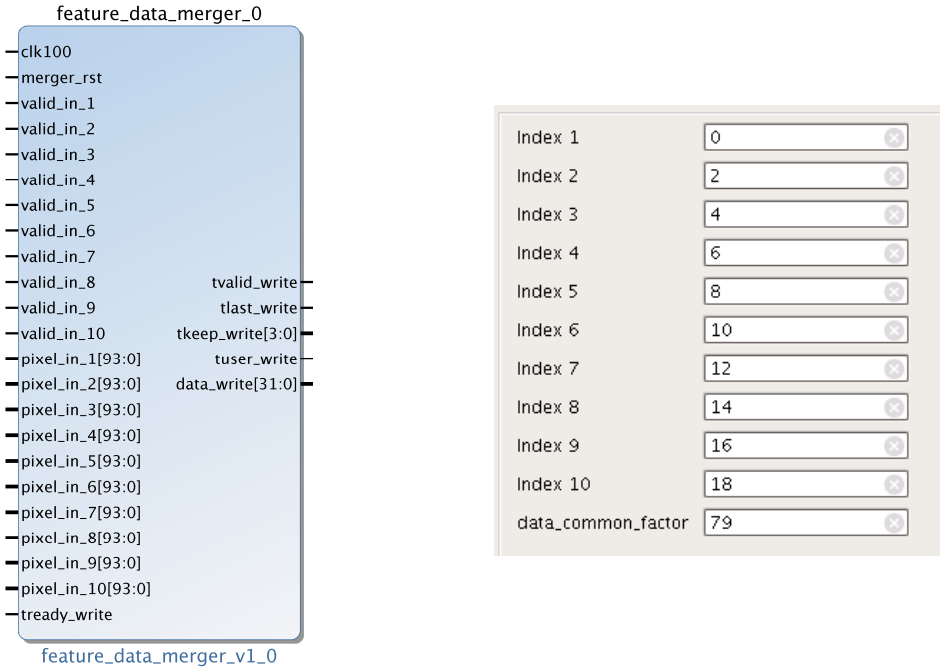


Figure 5.34: The packaged IP and configurable parameters of feature data merger module

In Figure 5.34, “valid_in_*” and “pixel_in_*” are the input ports of feature data for different scalers. And “*_write” are the AXI-bus compatible output ports that communicate with the VDMA. Besides, to generate correct “tlast” signal for VDMA data writing, the number of feature data to be transferred each time should be set up. Because this information is unknown before the practical application design, parameter “data_common_factor” which indicate the number of feature data to be transferred each time is provided. The “Index *” parameters in the configuration figure are the scale index added to each feature data. These changeable indexes should be match up with the indexes used in detection process. The resource utilization of this IP is listed in Table 5.17. To achieve data reformatting and sending to AXI-bus, a few LUTs and FFs are used. No BRAM or DSP is required in this module.

Site Type	Resource Used	Available
LUT	503	53200
FF	1114	106400
BRAM	0	140
DSP	0	220

Table 5.17: The resource utilization report of feature data merger IP on Zedboard

5.9 Classification

After all feature data are extracted from different scaled images, classification module receives these feature data and detects if there are objects in the image. During this process, sliding window and decision tree are employed.

In the design, this module is implemented on the ARM core instead of using programming logic. The main reasons of this choice are as follows:

- Firstly, for a 640×480 input image, the dimension of its feature image is 160×120 where each feature data is 94 bit. For pedestrian detection application, the size of sliding window is normally 128×64. Correspondingly, its sliding window in feature image is 32×16. To achieve object detection, minimum $33 \times 160 \times 94$-bit data should be buffered in BRAM. Thus, many BRAMs are required.

- Secondly, the sliding window based classification process should be performed under different image scales. To achieve object detection in real-time, the cycles for each classification can be calculated. The RGB565 data is obtained from the camera at 25MHz, therefore, the 24-bit pixel input clock is 12.5MHz. Because the dimension of feature image is 1/16 of the input and the processing clock is 100MHz. Therefore, 128(8×16) clock cycles are available for each sliding window based classification. Nevertheless, assuming the classification is required to repeat for eight different scales. Then, each classification under each scale should be done in 16(128/8) clock cycles. For a 2048 2-depth decision tree, minimum 128(2048/16) 2-depth decision tree should be calculated each cycle. Due to the fact that in 2-depth decision tree, the second layer conditional judgement is related to the result of the first layer conditional judgement result, it is impossible to finish a 2-depth decision tree in one cycle. Therefore, minimum $256(2048/16 * 2)$ times conditional statements should be performed each cycle. Besides, data fetching and decision tree results summarizing also consume a few clock cycles. Hence, in reality, 512 or more times of conditional judgements should

be performed each clock cycle. These conditional statements also utilizes a huge amount of resources on FPGA.

- Besides, the conditional statements cannot be dramatically speeded up by FPGA hardware implementation, especially the 2-depth decision tree.

Considering these reasons, classification module is implemented on the ARM core by C++. The data input of this module is feature data that stored in DDR3 memory. Because the feature data that feature data merger IP sent to DDR are out-of-order, the features in classification module should be reordered to guarantee that the features are grouped by their scale index. Then, padding is added to some scaled images. After padding is done, a sliding window based classification process is performed for the feature data of each scale. In the end, the location of detected objects are saved back to DDR3 memory. The inner structure of classification module on ARM is shown in Figure 5.35.

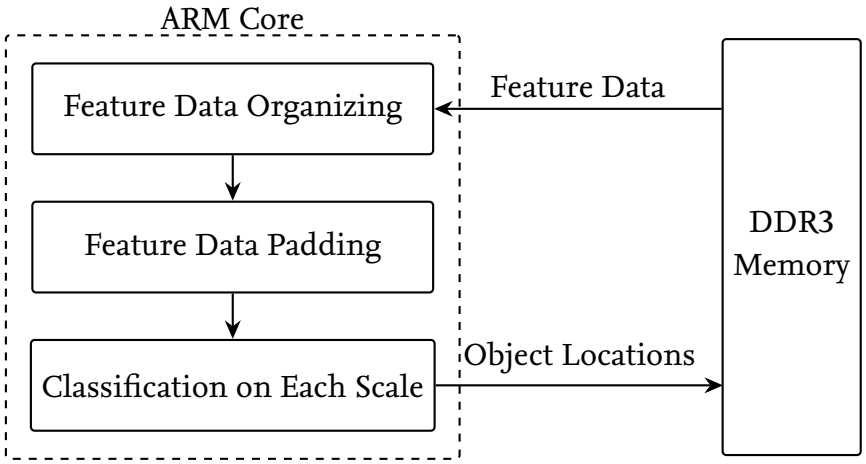


Figure 5.35: The block view of classification module

5.9.1 Feature Data Organizing

As mentioned in Section 5.8, the latencies of the feature calculation modules and feature scaler modules are all different. Therefore, the data that feature data merger sent to DDR3 memory via VDMA is out-of-order. However, the "scale index" that feature data merger IP added to each scale image guarantees the feature data still distinguishable. Hence, the first task of classification module is to categorize the feature data by "scale index". Assuming that features from 10 scaled images are extracted, after the categorization, 10 scaled feature image should also be obtained.

Worth to note that the "scale index" used in this section is exactly the same as the parameters that we set up in feature data merger IP. Moreover, the quantity

of each scale's feature data can also be calculated. Comparing this quantity to the number in theory by approximate computing. If the numbers are correct for each scale, the signal to start the second step of classification is generated.

5.9.2 Feature Data Padding

After feature data organizing, all feature data are settled. As mentioned in Section 2.x, during the object training phase, all training pictures are padded. For instance, for pedestrian detection, the size of pedestrians in training image is 100×41, but they are padded to 128×64 during training. Therefore, to detect the objects near image borders, the extracted feature image should also be padded. In this work, 3 pixels are padded to the left and right border of each scaled feature image, and 4 pixels are padded to the upper and lower border of each scaled feature image.

5.9.3 Classification on Each Scale

When the feature data are categorized and padded, a sliding window based classification can be performed for each scaled feature image. During this process, parameters obtained from the training phase are used. These parameters include:

- The size of the sliding window
- The number of employed 2-depth decision trees
- The threshold of each node on each 2-depth decision tree
- The confidence value of each node.

Although floating point operation is supported on the ARM process, some of the parameters obtained from Matlab are very small. To keep the data precision via computing and speedup calculation precess, all the parameters are left-shifted 8-bit in Matlab. The amplified parameters are then defined as the arrays that can be used during the classification. Figure 5.36 illustrates an example of 2048 decision tree based classification process for pedestrian detection.

From this figure, it can be seen that each decision tree contains 2 layers. With two times comparison, the confidence value can be obtained from each decision tree. Thereafter, the final confidence value of current sliding window is calculated by summarizing all the confidence values from each decision tree.

In this work, to speed up the classification process, a confidence threshold param-

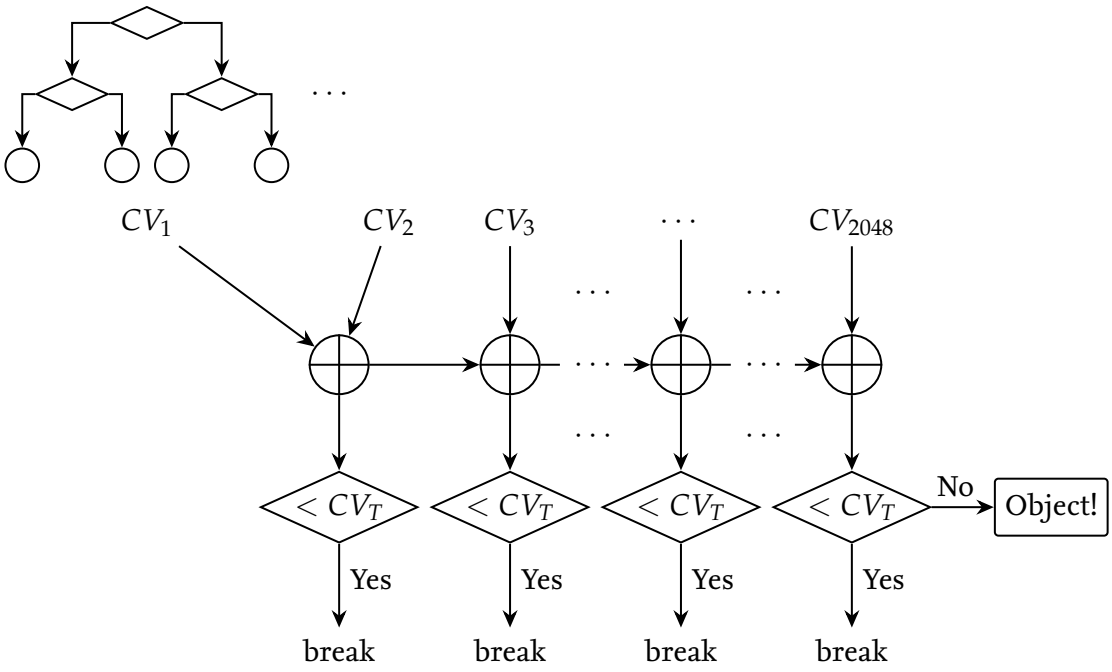


Figure 5.36: An illustration of the decision tree-based classification process

eter CV_T is added. When the sum of the first few confidence values is smaller than the threshold CV_T, this sliding window is classified as no object inside. The reason that this threshold works is, for an input image, the features of blank, black, etc. area are quite different compared to the features of an object. Thus, the classification process can break quite early when the sliding window is in these areas. After experimental test for many times, the results show that -256 is a proper threshold for pedestrian detection and traffic sign detection applications. Besides, from the speed testing results, we can conclude that with the confidence threshold CV_T, the classification process of pedestrian detection is 16 times faster.

Although speed test is not performed for other object detection applications, however, since with the threshold CV_T, the comparison process can be stopped early for many sliding windows, it can be deduced that all object detection applications can be speed up with CV_T parameter.

In the end, when sliding window is performed for all scaled images, the location of detected objects will be saved to DDR3 memory again.

5.10 Candidate Selection

The job of candidate selection module is to obtain the classification results from classification module and remove the overlapped data. From Section 5.9, we know

that the classification results are the locations of all detected objects. However, in reality, many of these results are pointing to the same object, which is caused by the following reasons:

- Firstly, sliding window is used on feature data image in classification module. Because the sliding window is moving pixel-by-pixel, when an object is detected in one sliding window, it can also be detected easily in the slightly moved sliding window.
- Besides, to detect different sized objects, the classification process is executed under different image sizes. Nevertheless, for the neighboring image scales, the size of an object is not that much differentiated. Therefore, the same object can usually be detected under two or three image scales.

In this module, the results from classification IP serve as the candidates for further selection to generate the final detection results. In the design, three processing tasks are performed to these candidate data: location data sorting, data rescaling, and overlapped results removing. Location data sorting submodule reorders the location data by confidence value. Thereafter, to compute the overlap between these data, the reordered data are rescaled to the same resolution. In the end, the overlapped data with lower confidence value are removed. A brief inner structure of this module is shown in Figure 5.37.

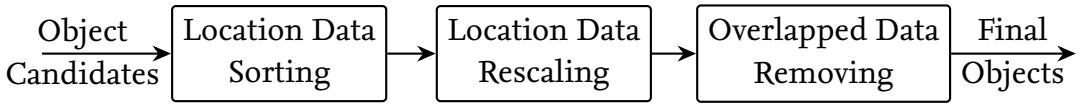


Figure 5.37: The brief overview of candidate selection module

5.10.1 Location Data Sorting

The location data transferred from classification module contains image scale index, the coordinates of upper-left corner of the object, width and height of the object, and the confidence value of this classification. For the ease of description, the confidence value of an object is denoted as C_v, and its corresponding other location information are summarized into Obj. Then these location data can be represented as $[Obj_1, C_{v_1}]$, ..., $[Obj_n, C_{v_n}]$. The task of location data sorting submodule is to sort these data by the confidence value C_v, as illustrated in Figure 5.38.

Herein $[C'_{v_1}, Obj'_1]$, ..., $[C'_{v_n}, Obj'_n]$ are the reordered object candidates and C'_{v_1} is

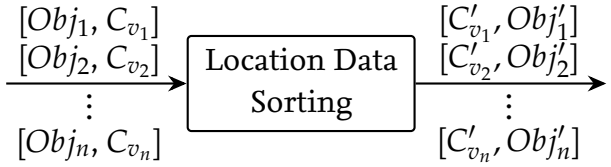


Figure 5.38: An illustration of location data sorting

detected object with highest confidence value.

5.10.2 Location Data Rescaling

Because the location data are detected from different image scales, in order to calculate the size of overlapped area, all location results should be rescaled to the same scale. In the design, these data are all rescaled to their values in the original image. During the rescaling process, the scaling ratio of approximate computing s_i, the real scaling ratios for height and width s_{w_i} and s_{h_i} are used. If we denote the location information Obj_i as $[x_i, y_i, w_i, h_i]$, the final location of each detected data in 640×480 can be obtained by:

$$x_{i_{final}} = \frac{x_i + x_{shift}}{s_{w_i}}, \quad y_{i_{final}} = \frac{y_i + y_{shift}}{s_{h_i}}, \quad \text{with} \quad i \in [1, n], \tag{5.21}$$

where x_{shift} and y_{shift} being the parameters to remove image padding that introduced in classification module. The corresponding width and height of the object in 640×480 can be calculated by:

$$w_{i_{final}} = \frac{w_i}{s_i}, \quad h_{i_{final}} = \frac{h_i}{s_i}, \quad \text{with} \ \ i \in [1, n]. \tag{5.22}$$

5.10.3 Overlapped Results Removing

After the location data are sorted and rescaled, the overlapping rate between different location data can be computed. Assuming two location data after rescaling are $[x_{i_{final}}, y_{i_{final}}, w_{i_{final}},$
$h_{i_{final}}]$ and $[x_{j_{final}}, y_{j_{final}}, w_{j_{final}}, h_{j_{final}}]$, the overlapping in x and y direction is equal to:

$$\begin{aligned} w_{ij_{overlap}} &= \max((\min(x_{i_{final}} + w_{i_{final}}, \ x_{j_{final}} + w_{j_{final}}) - \max(x_{i_{final}}, x_{j_{final}})), 0) \\ h_{ij_{overlap}} &= \max((\min(y_{i_{final}} + h_{i_{final}}, \ y_{j_{final}} + h_{j_{final}}) - \max(y_{i_{final}}, y_{j_{final}})), 0), \end{aligned} \tag{5.23}$$

where $w_{ij_{overlap}}$ and $h_{ij_{overlap}}$ are the overlapped part in x and y direction, respectively. Then, the overlapping area $S_{ij_{overlap}}$ can be calculated by:

$$S_{ij_{overlap}} = w_{ij_{overlap}} \cdot h_{ij_{overlap}} \tag{5.24}$$

In the design, two different overlapping removing approaches are implemented: overlapping for one and overlapping for all.

Overlapping for One

The first approach is to calculate the overlapping rate for each location and select the bigger one as the final overlapping rate, which can be calculated by the following equation:

$$R_{ij_{overlap}} = \max\left(\frac{S_{ij_{overlap}}}{w_{i_{final}} \times h_{i_{final}}}, \ \frac{S_{ij_{overlap}}}{w_{j_{final}} \times h_{j_{final}}}\right), \tag{5.25}$$

where $R_{ij_{overlap}}$ is the final overlapping rate. If $R_{ij_{overlap}}$ is larger than 50%, the detected result with lower confidence value will be removed.

Overlapping for All

The other approach that implemented in this IP is to calculate the ratio of overlapping area to the sum of two locations' area, which can be written as:

$$R_{ij_{overlap}} = \frac{S_{ij_{overlap}}}{w_{i_{final}} \times h_{i_{final}} + w_{j_{final}} \times h_{j_{final}} - S_{ij_{overlap}}} \tag{5.26}$$

When the overlapping rate $R_{ij_{overlap}}$ is larger than 50%, the detected result with lower confidence value will be deleted.

Figure 5.39 illustrates one scenario with two different overlapping removing approaches. The left figure of Figure 5.39 employes the first approach. Because the overlapping rate is bigger than 50% and the confidence value of the smaller window is lower, it is deleted. On the other hand, the right figure shows the result of "Overlapping for All" approach. Because the overlapping rate is smaller than 50%, both results are reserved.

Both overlapping removing approaches can be useful under different scenarios. For instance, in pedestrian detection application, when the front pedestrian is bigger in screen, the human which is farther away can also be detected by overlapping for all approach. Therefore, the second approach can effectively reduce the detection miss rate. Meanwhile, for traffic sign detection application, the situation of

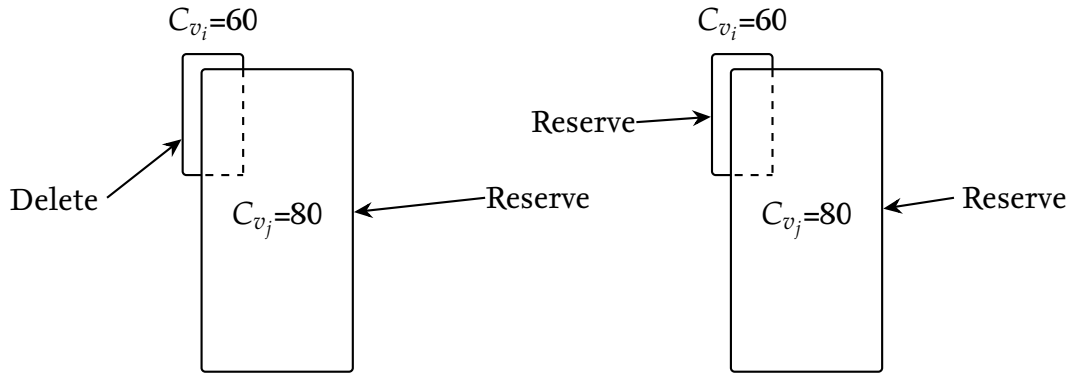


Figure 5.39: An illustration of two different overlapping data removing approaches

front object blocks being farther away object never happens, using overlapping for one method can remove some false positive results.

IP Packaging and Resource Utilization Analysis

Figure 5.40 is the configuration interface of candidate selection IP. It can be seen

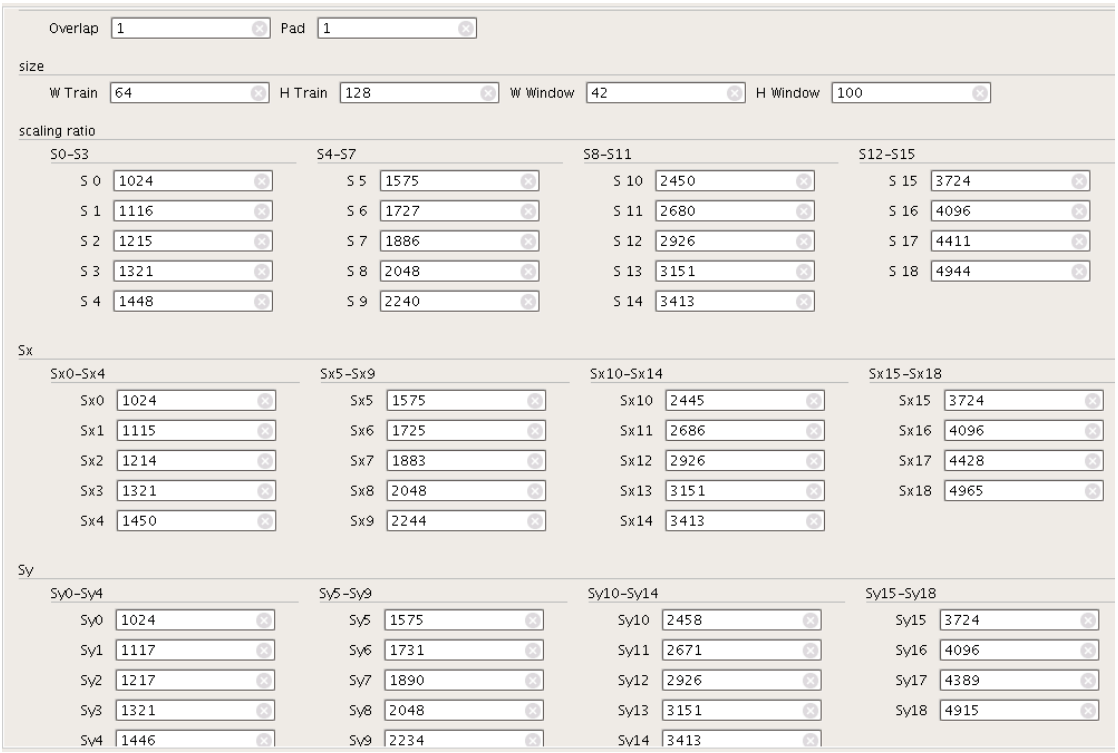


Figure 5.40: The configurable parameters of candidate selection module

that many parameters are configurable in this IP. The parameter “Overlap” indicates which overlap removing approach is selected:

- '0': Overlapping for One approach,
- '1': Overlapping for All approach.

The parameter "Pad" indicates if padding is added to the images under different scales. Without correct "Pad" configuration, the objects can still be detected, however, the displacement of object location is shown on screen. "W Train", "H Train", "W Window" and "H Window" are the parameters to indicate if padding is used during the training phase. When used, another correction is performed for rectangle drawing module. Because the location data are rescaled in this module, scaling ratio of approximate computing, real scaling ratios for width and height are also configured by parameters "S0-S18", "Sx0-Sx18", and "Sy0-Sy18". All these scaling parameters are calculated from MATLAB.

The packaged IP of candidate selection module is shown in Figure 5.41. An AXI compatible version interface is provided for the data input. However, this IP will still work without AXI bus, if "tvalid_in" and "tdata_in" signals are given. The module works under 25MHz clock rate and "module_rst" is provided as the reset and frame synchronization port. The processing results are 48-bit output, which includes 10-bit x-, y-coordinates, 10-bit width, 10-bit height, and 8-bit confidence value. Besides, one extra signal "location_rst" is provided to indicate that no object is detected in current image. This signal is useful for drawing module when refresh the displaying image.

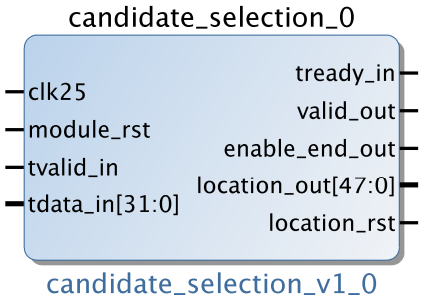


Figure 5.41: The packaged Ip of candidate selection module

Site Type	Resource Used	Available
LUT	901	53200
FF	1196	106400
BRAM	1	140
DSP	8	220

Figure 5.42: The resource utilization report of candidate selection IP on Zedboard

Finally, the resource utilization report of candidate selection IP is given in Table 5.42. Because many comparison operations are existed in the location data sorting submodule, many LUTs and flip-flops are used. Besides, one Bram is used to store all location candidates. Moreover, multiplication operations are exist in location data rescaling and overlapped data removing submodule, thus, 8 DSPs are utilized.

5.11 Drawing

Drawing module implements rectangle drawing on screen. After the final detected results are obtained, drawing module will draw a rectangle around each object. The data input of this module contains original image data and object locations. All these data are read from DDR3 memory via VDMA. However, due to the fact that storing locations of these data are different, two VDMAs are employed to read these data.

In this module, for each pixel read in, its coordinates are compared to the rectangle border of the object, when this pixel is on the border of one object, the color of this pixel will be changed to red. When there are more objects in one image, this comparison process will repeat many times.

To achieve this, a 5-stage pipeline architecture is designed in this module. The latency of the pipeline architecture is 17 clock cycles. In this design, maximum 30 objects can be drawn on screen, which is the limitation of the 5-stage pipeline structure. With a more stage pipeline architecture, more comparison operations can be performed. Correspondingly, more rectangles can be drawn. However, in reality, 30 objects are already enough for most applications.

IP Packaging and Resource Utilization Analysis

The packaged IP and resource utilization report of this IP are shown in Figure 5.43 and Table 5.44, respectively. Two input clocks are used in this IP: 25MHz and 100MHz. 25MHz clock is used to read in pixel data and 100MHz is the clock

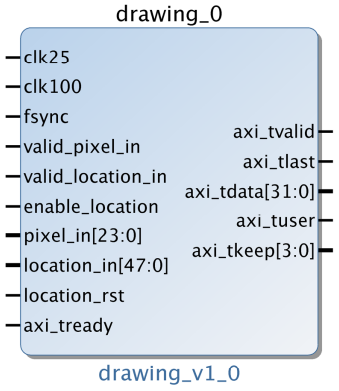


Figure 5.43: The packaged IP of drawing module

Site Type	Resource Used	Available
LUT	1679	53200
FF	940	106400
BRAM	0	140
DSP	0	220

Figure 5.44: The resource utilization report of the drawing IP on Zedboard

rate of processing blocks. Image pixels and final locations of detected objects are read from DDR3 memory via VDMA. “axi_*” ports are the output of the drawing

module, which transfers the image data with rectangles also to the DDR3 memory via another VDMA. Because many comparison operations exist in this module, many LUTs and FFs are consumed.

5.12 Display

The task of display module is to read pixel data from BRAM or DDR3 memory and then display them on the screen via VGA or HDMI interface. In this section, two IP cores that can display images on monitor are created: VGA IP and HDMI IP.

5.12.1 VGA

On Zedboard, the VGA interface only supports RGB 444 format. Therefore, the output of VGA IP is 12 bit, where 4 bit for red, green, and blue channel each. To achieve different resolution images displaying via VGA port, some configurable parameters are provided. The view of packaged VGA IP and its configurable interface is shown in Figure 5.45.

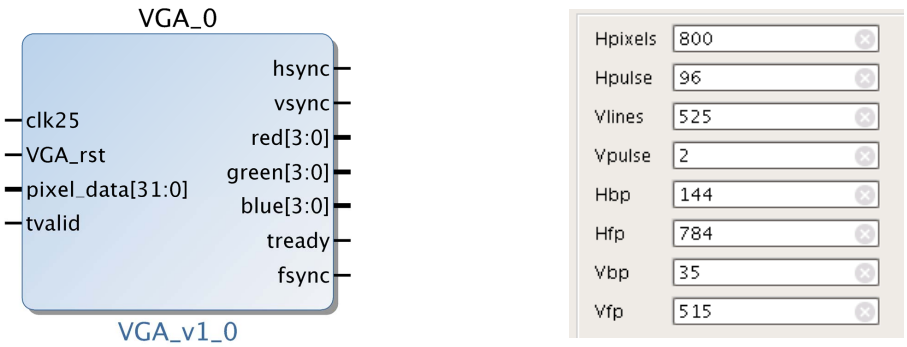


Figure 5.45: The packaged IP of VGA module

The function of each parameter in Figure 5.45 is listed in Table 5.18. By configuring these parameters, three resolutions are verified and tested by using the VGA IP. The configurations of these supporting resolutions are listed in Table 5.19, which is set up according to the industry standard of VGA signal timing [85]. The 640×480 VGA configuration can be used to display camera output image. And stereo images from two OV7670 cameras can be displayed with 1280×960 configuration.

Parameter	Function
Hpulse	The length of hsync pulse
Hpixels	The number of horizontal pixels per line
Vlines	The number of vertical lines per frame
Vpulse	The length of vsync pulse
Hbp	The end of horizontal back porch
Hfp	The beginning of horizontal front porch
Vbp	The end of vertical back porch
Vfp	The beginning of vertical front porch

Table 5.18: The explanation of configurable parameters of the VGA IP

Resolution	Paramemters							
	Hpixels	Vlines	Hpulse	Vpulse	Hbp	Hfp	Vbp	Vfp
640×480 @$60Hz$	800	525	96	2	144	784	35	515
800×600 @$60Hz$	1056	628	128	4	216	1016	27	627
1280×960 @$60Hz$	1712	994	136	3	352	1632	33	993

Table 5.19: Verified supporting resolutions of the VGA IP

5.12.2 HDMI

As mentioned in Section 5.12.1, the VGA interface on Zedboard only supports RGB444 format. To display the image data with better quality, another display IP HDMI is created.

Figure 5.46 illustrates a brief overview of the inner structure of HDMI IP. Because the HDMI interface use YUV color space instead of RGB color space. Therefore, a color space conversion from RGB to YUV is performed. Thereafter, the YUV data are reformed to the HDMI interface required data format.

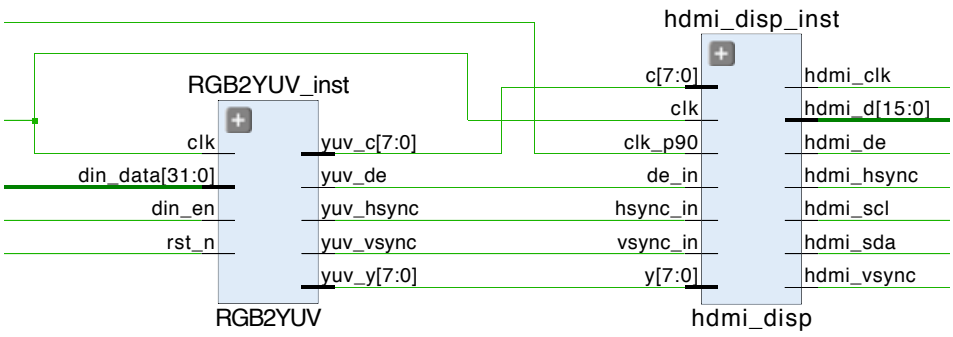


Figure 5.46: The inner structure of the HDMI module

For the RGB to YUV color space conversion, if the color values of point P in RGB color space is denoted as (R_i, G_i, B_i) and its corresponding color channel values in YUV color space as (Y_i, U_i, V_i), the conversion can be expressed by:

$$\begin{aligned} Y_i &= \frac{8432 * R + 1645 * G + 3176 * B}{32768} + 16; \\ U_i &= \frac{-4818 * R - 9527 * G + 14345 * B}{32768} + 128; \\ V_i &= \frac{14345 * R - 12045 * G + 2300 * B}{32768} + 128; \end{aligned} \tag{5.27}$$

In Equation (5.27), the conversion matrix is already rewritten to integer format by left-shifting all parameters bz 15 bits. (Note that parts of the RGB to YUV verilog code are referred to Mike Field's work in [86].)

The packaged IP of HDMI module is shown in Figure in 5.47. Two input clocks are used in this module: "clk" is the normal clock for data processing, and "clk_p90", which has a 90 degrees' phase difference with "clk", is used to generate the "hdmi_clk" signal used in HDMI interface.

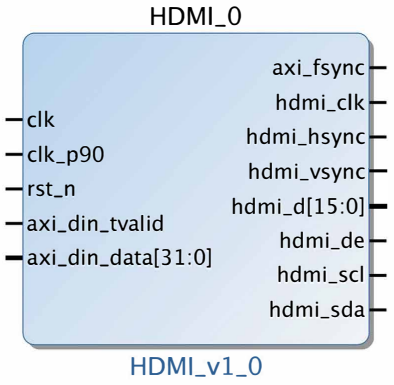


Figure 5.47: The packaged IP of HDMI module

Moreover, the data passed into this IP can be either from normal IP core or from AXI-bus. The "hdmi_*" ports are the pins that connected to the Zedboard.

The resource utilization of VGA and HDMI module is listed in Table 5.20. It is evident that only few LUTs and FFs are used to generate VGA IP. However, relatively more resources are used for HDMI IP generation. Because multiplication operations are performed in color space conversion, therefore, six DSPs are employed in the HDMI module.

Site Type	Resource Used		Available
	VGA Module	HDMI Module	
LUT	38	122	53200
FF	22	246	106400
BRAM	0	0.5	140
DSP	0	6	220

Table 5.20: The resource utilization report of VGA and HDMI module on Zedboard

5.13 Summary

In this chapter, many IP cores that can be used for different object detection applications are designed and implemented. The detailed design process and configurable parameters of each IP are described in detail. Furthermore, the resource utilization report of each IP is given and analyzed at the end of each module.

Chapter 6

Design of Object Detection Systems with IP-Toolbox

In Chapter 5, IP cores that correspond to each block of the generalized object detection framework are designed, implemented, and tested. Based on these IP cores, in this chapter, different object detection systems (e.g., pedestrian detection system, traffic sign detection system) are designed and verified on Zedboard.

6.1 Object Detection Platform

For a vision-based object detection system, camera is the essential device that ordinarily serves as the input of the system. In this work, video camera OV7670 [72] is selected and utilized. Moreover, Zedboard is employed as the "brain" of the detection system, which processes the camera data, detects objects, and generates output to display. Besides, a monitor is required to display the video output. In this section, by connecting the cameras, Zedboard, and a monitor, the whole object detection platform is created, as can be seen in Figure 6.1.

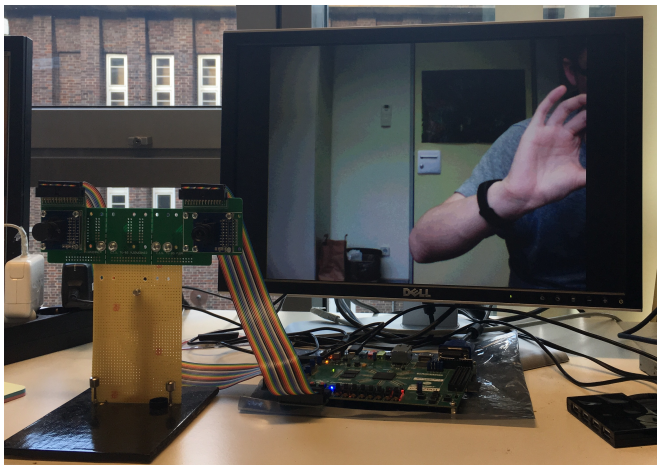


Figure 6.1: An illustration of the object detection platfrom

Since the camera OV7670 can not be connected to the Zedboard directly, PCB boards, which connects the camera and Zedboard with Pmod interface, are designed and printed. Figure 6.2 illustrates the two PCB boards that are eventually employed: one board plugs into the Zedboard side and the other fixes the camera. The flat ribbon cable is utilized to connects two PCB boards.

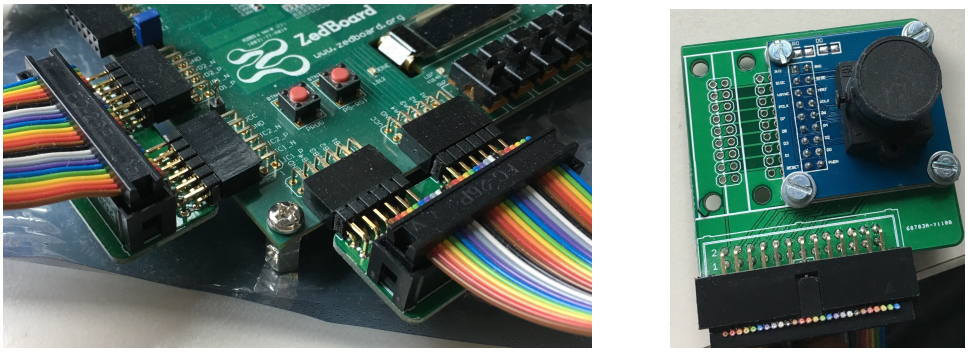


Figure 6.2: The PCB boards that connect the OV7670 camera and the Zedboard

As mentioned in the generalized object detection framework (Figure 4.1), in order to compute depth information of detected objects, stereo cameras are employed as the video input. However, the stereo-camera calibration and rectification process is valid only with the premise that the distance between two cameras and camera focus length are fixed. Therefore, a stable platform, as shown in Figure 6.3, is manually created. In consequence, the distance (baseline) between two soldered cameras are fixed and can be measured.

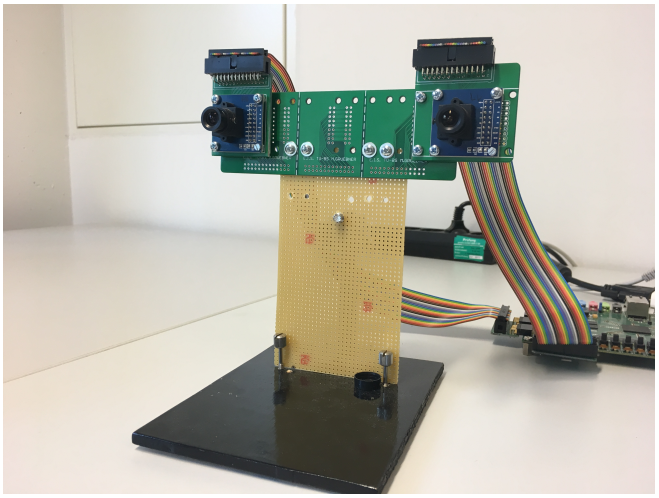


Figure 6.3: An illustration of the stereo camera platform

For the video output, either VGA or HDMI interface of the Zedboard can be used since both VGA IP and HDMI IP are designed in Section 5.12. Considering the

programming logic resources that used by the VGA IP and the HDMI IP, VGA interface is eventually employed by the pedestrian detection system and traffic sign detection system.

6.2 Design of Pedestrian Detection System on a SoC-FPGA Platform

With the IP cores that designed in Chapter 5 and the platform that created in Section 6.1, the pedestrian detection system is designed, implemented, and tested in this section.

According to the fast feature pyramid theory that introduced in Section 4.1.2, for an input image with the resolution of 640×480 and the size of the sliding window being 128×64, the feature calculation process is required to execute three times where each time extracts the feature data from an input image with the resolution of 640×480, 320×240, and 160×120, respectively. Besides, 16 times of feature scaling are required to obtain the feature data under different image scales. However, the synthesis report of Vivado shows that the DSPs and BRAMs on Zedboard are not enough to generate the system with 19 layers of feature data. To ensure the designed system can be synthesized, in the end, 10 layers of feature data are employed, which includes 3 layers of calculated feature data and 7 layers of scaled feature data. The block design of the feature extraction part of the system is shown in Figure 6.4.

In Figure 6.4, two image scaler IPs are employed to generate the input image with the resolution of 320×240 and 160×120. When the feature data are calculated by each feature calculation IP, several feature scaler IPs are connected to each feature calculation IP. Each feature scaler IP is configured to compute the features of one scaled image by the bilinear interpolation method. All of the feature calculation IPs and feature scaler IPs are configured by the results that MATLAB calculated according to fast feature pyramid theory. When the feature data of ten scaled images are obtained, the feature data merger IP is used to merge and transfer all of the feature data to the DDR3 memory via a Video Direct Memory Access (VDMA).

Because the size of the sliding window utilized in this design is 128×64, to achieve real-time object detection, minimum 128 lines of pixel data are required to be buffered in BRAM. Considering the limited resources on Zedboard, the classification module is implemented in the ARM core. Besides, due to the time-delay introduced by data buffering of feature calculation IP, feature scaler IP, image scaler IP, and drawing IP, the captured raw image data and some intermediate results

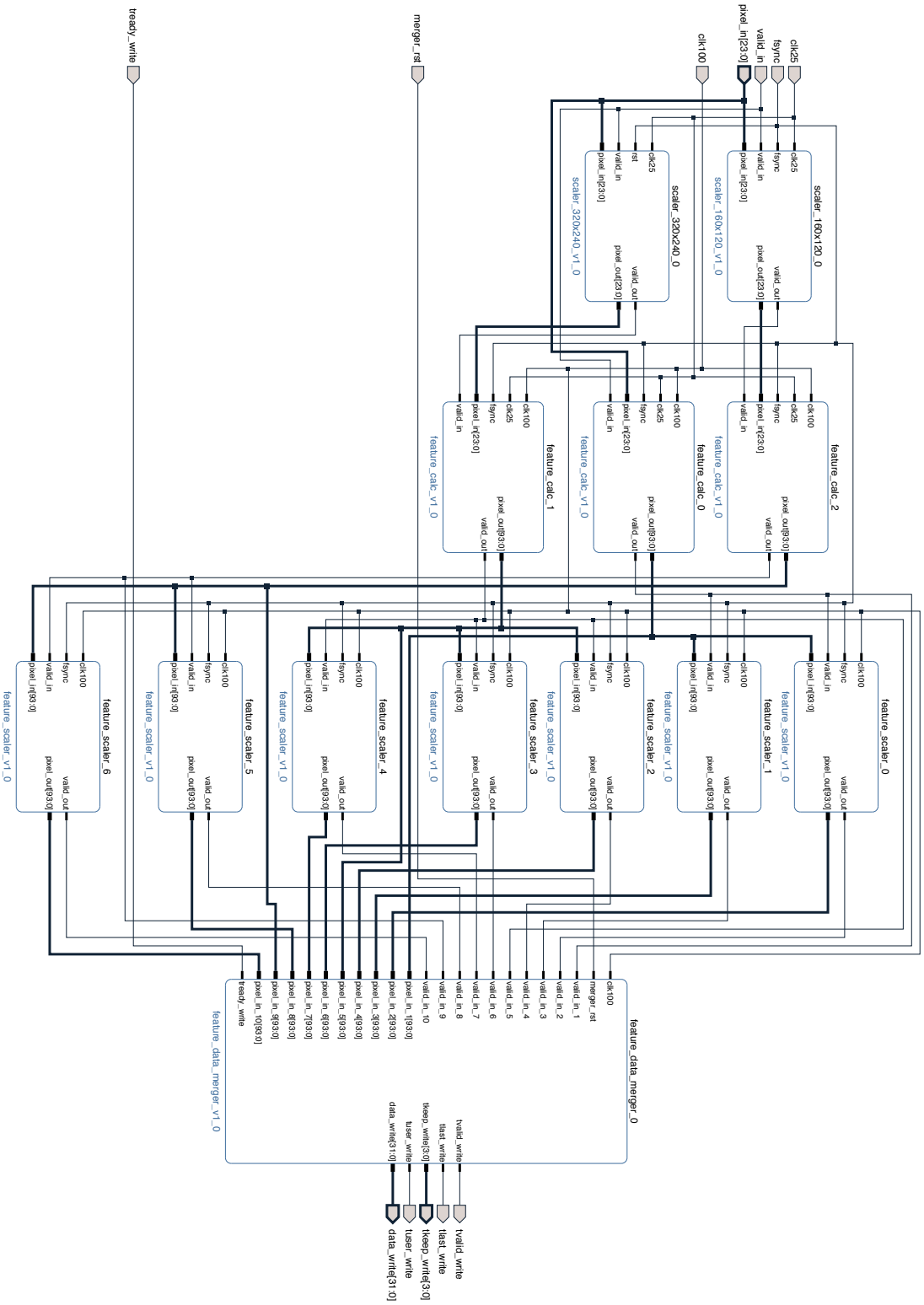


Figure 6.4: The block design of feature extraction

are also required to be saved. In the design, these data are all stored in the DDR3 memory and multiple AXI VDMAs are employed to transfer the data between IP blocks and the DDR3 memory.

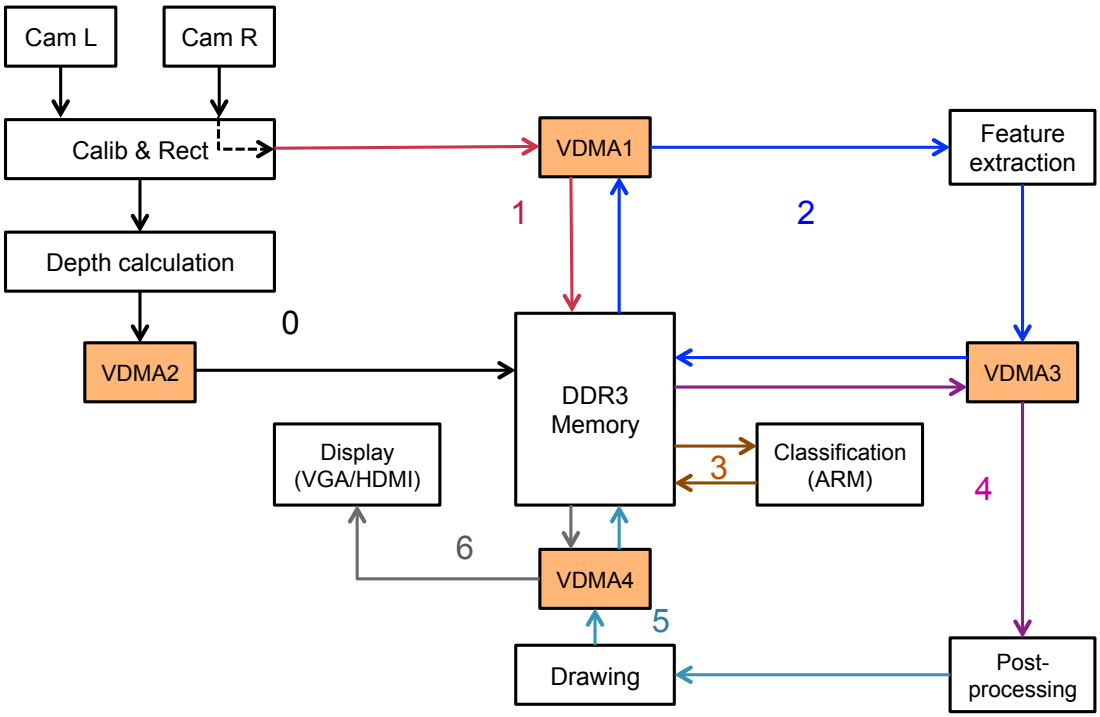


Figure 6.5: The flow of streaming data in the pedestrian detection system

The flow of streaming data in the system can be divided into six steps, which is shown in Figure 6.5:

- Firstly, images are captured by the camera module and then stored to the DDR3 memory via configuring the write channel of VDMA1. To guarantee the fluency of the streaming data, three frames of captured image are buffered.
- Secondly, the feature extraction module reads the image data from DDR3 memory via configuring the read channel of VDMA1. Then, features are extracted and stored to the DDR3 memory by the write channel of VDMA3.
- When the feature data is ready, the classification module that inside the ARM core starts. The classification module reads the feature data from DDR3 memory and sends the detection results back to the DDR3 memory.
- Fourthly, the information of detected objects are transferred to candidate selection module (which is named as post-processing in FIgure 6.5) via VDMA3

read channel. In candidate selection module, duplicated candidates are removed and the final coordinate information of detected objects are passed to the drawing module.

- The drawing module merges the pixel data of original image and the coordinate information of detected objects. Then, the original image with rectangle results are written back to the DDR3 memory via VDMA4 write channel.
- Meanwhile, the stereo image data are passed to stereo calibration module, image rectification module and depth calculation module. Then, the depth data are stored to the DDR memory via the write channel of VDMA2. The connections of stereo camera IP cores are shown in Figure 6.6. Worth mentioning that due to the computation complexity of the depth calculation process, the resolution of obtained depth data is only 160×120.
- Finally, the image data with rectangle results are read out from the DDR3 memory via VDMA4 read channel and displayed on a monitor.

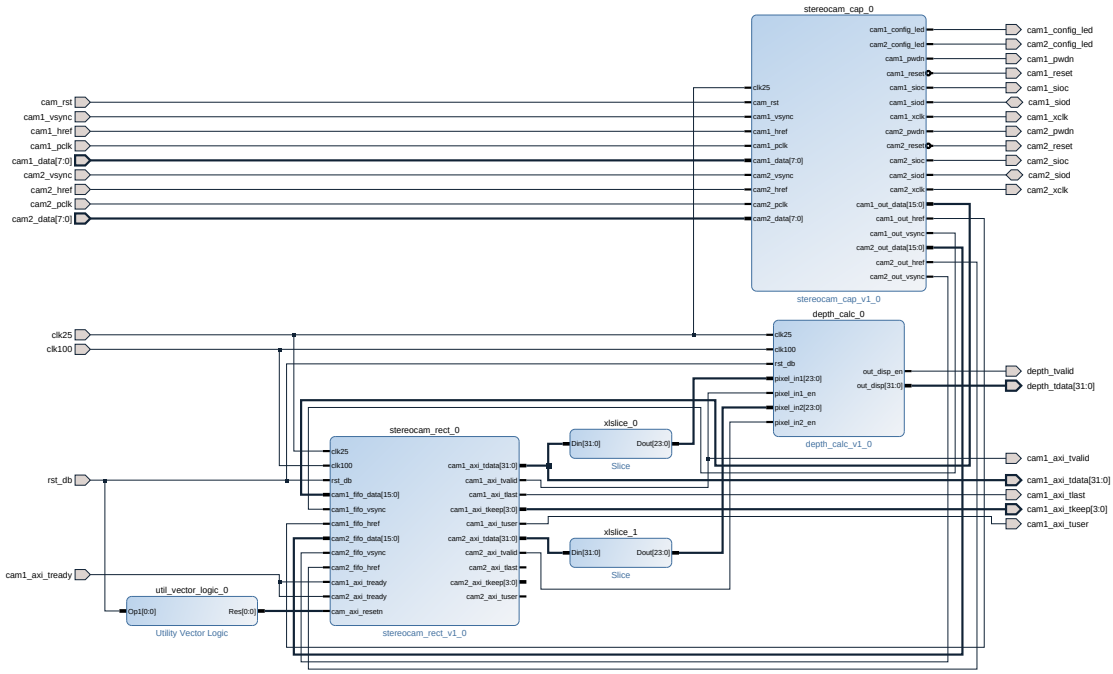


Figure 6.6: The block design hierarchy of depth calculation

When the bitstream file is successfully generated by the block design and the platform is correctly connected, the pedestrian detection system is powered on and tested. Some results of our pedestrian detection system are illustrated in Figure 6.7 (right side). Figure 6.7 (left side) provides corresponding results from the MATLAB version of the pedestrian detection algorithm.

Figure 6.7: The difference of detected results between MATLAB version pedestrian detection algorithm and the designed system

Detection Accuracy

By comparing the results of MATLAB version algorithm and of our pedestrian detection system that shown in Figure 6.7, we can notice that differences exist between them. For a real-time detection system, it is difficult to measure the detection accuracy or false positive rate. However, the confidence value provides another degree of evaluating the differences between the results from MATLAB version pedestrian detection algorithm and the results detected by our designed system. The higher the confidence value is, more confidence the detection object ture-positive is.

For more detailed information about confidence value, please refer to Section 4.1.3. For the example that illustrated in Figure 6.7 (upper), the confidence value of each pedestrian is 68.38 and 64.57, respectively. Meanwhile, for the same picture, the confidence values obtained by the Zedboard are 57 and 55, respectively. For this example, slightly difference exists between the confidence values, however, both objects are correctly detected.

From the second example illustrated in Figure 6.7 (lower), it is evident that more differences exist between the detected results. In this case, more objects are de-

tected by the MATLAB version algorithm where one more false positive object is also detected. Meanwhile, one pedestrian with low confidence value is missed by our designed system. Therefore, slightly performance decline may exist in the designed pedestrian detection system. However, by comparing the confidence value that calculated from MATLAB and the system, the results show that this performance decline is not sever. The possible reasons behind this difference can be as follows:

- Firstly, look-up table (LUT) is employed in the feature calculation module to replace the division and square root operations. Precision loss is introduced by employing the LUTs.
- Secondly, the left-shifting is used while converting the floating point operations to the integer operation. The bit-width that shifted and the bit-width that kept may also lead to precision loss.
- Besides, some intermediate results are required to be stored in the BRAM. However, due to the limitation of BRAM resource on Zedboard, limited bits are kept for these intermediate results. Hence, the error between the real value and the value obtained by our IP cores is also introduced.

It is worth noting that in the designed pedestrian detection system, the confidence value information is obtained via the UART interface. It is only used while debugging the system.

Resource Utilization

Owing to a large number of resources are utilized by StereoCam rectification IP and depth calculation IP, only one feature calculation IP and two feature scaler IPs can be added to the system. Therefore, the feature data are only extracted under three image scales. The depth data of detected objects are passed via the UART of Zedboard.

The resource utilization report of the final implementation of pedestrian detection system on Zedboard is given in Table 6.1. From this table, we can conclude that around 65% of LUTs and DSPs on Zedboard are used by the pedestrian detection system without depth calculation. Due to the differences of the configurations of the feature scaler IP and feature calculation IP, the resources used to implement each feature scaler IP and feature calculation Ip are slightly different. Because the utilization of Xilinx Parameterized Macros (XPM) in the feature scaler IP, configurable parameter of BRAM size is provided. Therefore, the feature scalers that employed to process the data with 320×240 and 160×120 resolution, a num-

Site Type	LUT	FF	BRAM	DSP
axi_mem_intercon	1146	1312	0	0
axi_mem_intercon_1	1149	1312	0	0
axi_mem_intercon_2	1146	1312	0	0
axi_vdma_0	2041	3185	2.5	0
axi_vdma_1	2053	3257	2.5	0
axi_vdma_2	2035	3189	2.5	0
feature_scaler_0	608	857	3	15
feature_scaler_1	614	857	3	15
feature_scaler_2	739	880	1.5	14
feature_scaler_3	615	853	1.5	15
feature_scaler_4	619	853	1.5	15
feature_scaler_5	775	876	1.5	14
feature_scaler_6	610	849	1.5	15
drawing	2509	1287	0	0
feature_calc	3720	4747	31	12
feature_calc_1	3747	4747	31	12
feature_calc_2	3654	4733	28.5	12
candidate_selection	1421	2003	1	8
camera_capture	66	153	0.5	0
feature_data_merger	504	1114	0	0
processing_system7	0	0	0	0
processing_system7_axi_periph	1873	2277	0	0
320x240_image_scaler	203	295	2	0
160x120_image_scaler	212	307	4	0
rst_processing_system7	19	29	0	0
vga_0	38	22	0	0
Sum	32116	41306	119	147
Available	53200	106400	140	220

Table 6.1: The resource utilization report of pedestrian detection system on Zedboard

ber of 1.5 BRAM is used. Besides, it is worth noting that “axi_mem_intercon*”, “rst_processing_system7”, and “processing_system7_axi_periph” are the IP cores that added by the Vivado software to achieve system connection between ARM, VDMA and AXI interfaces.

The resource utilization report of the pedestrian detection system with depth calculation function is listed in Table 6.2. Because only one feature calculation IP and two feature scaler IP cores are employed in the version system, much less

DSPs are utilized. As can be seen in Table 6.2, 98.5 of 140 BRAMs are used, which seems that more image scales can still fit into the system. However, because each feature calculation IP should be connected to multiple number of feature scaler IP, the BRAMs left is not large enough to achieve that. Therefore, in the end, only one feature calculation module and two feature scalers are added in this version system. In the other words, BRAM is the bottleneck that actually limits the implementation of a bigger pedestrian detection system on the Zedboard platform.

Site Type	LUT	FF	BRAM	DSP
axi_mem_intercon	1147	1314	0	0
axi_mem_intercon_1	1149	1314	0	0
axi_mem_intercon_2	1146	1314	0	0
axi_vdma_0	2041	3184	2.5	0
axi_vdma_1	2053	3257	2.5	0
axi_vdma_2	2023	3177	2.5	0
feature_scaler_0	608	857	3	15
feature_scaler_1	614	857	3	15
drawing	2523	1267	0	0
feature_calc	3740	4747	31	12
candidate_selection	1421	2003	1	8
stereocam_capture	391	523	0	6
feature_data_merger	479	1078	0	0
stereocam_rect	3661	8706	39	44
depth_calc	5301	11788	14	0
processing_system7	0	0	0	0
processing_system7_axi_periph	1554	1653	0	0
rst_processing_system7	17	29	0	0
vga_0	38	22	0	0
Sum	29772	47597	98.5	100
Available	53200	106400	140	220

Table 6.2: The resource utilization report of pedestrian detection (with depth calculation) system on Zedboard

Running Speed

With respect to the running speed of the system, because of the pipeline structure is designed for each IP core that introduced in Chapter 6, 80 frame per second (fps) can be achieved for the IP cores that designed. The detailed information of the running speed of each IP core, please refer to Chapter 6. However, the processing speed of the classification module on the ARM core is difficult to predict, which

is caused by the conditional judgement strategy that employed in the boosting decision tree. Nevertheless, through classification module optimization and multiple times of on-board testing, we conclude that when there are different number of pedestrians in one image frame, 25 to 40 fps (real-time) pedestrian detection can be achieved. When there are more than 20 objects in the image, the rectangle marker of detected objects may be drawn in the next frame of the image. Comparing to the running speed of the framework on MATLAB, which requires ($\approx$) 0.4s to process each image, the detection speed is 10-16 times faster.

Working Distance

While testing the detection accuracy, the working distance of the system is also measured. In this measurement, due to the difference of image scales that involved in the designed, the working distance of the system with and without depth calculation module are both evaluated. Figure 6.8 (left) illustrates the maximum size of a human, namely, 240×480, that can be detected by the non-depth version detection system. Correspondingly, the minimum working distance can be calculated, which equals to ($\approx$) 3.5m. The minimum size of a pedestrian detected by the system is 41×100, as shown in Figure 6.8 (right). Thus, the corresponding maximum working distance of the system is ($\approx$) 21m. In other words, under good lighting conditions, the working distance of the pedestrian detection system is 3.5m $\sim$ 21m.

Figure 6.8: An illustration of the minimum and maximam working distance of the pedestrian detection system

Similarly, the working distance of the pedestrian detection system with depth calculation is also measured. Referring to Table 6.2, because the object detection is only processed under three image scales, the working distance of this system is 3.5m $\sim$ 6m.

6.3 Design of Traffic Sign Detection System on a SoC-FPGA Platform

In this section, the traffic sign detection system is designed based on the IP cores that implemented in Chapter 5. In fact, the hardware design of the traffic sign detection system is similar to the pedestrian detection system that designed in Section 6.2, except the following differences:

- Firstly, the configurations of feature extraction IPs are different. Refer to the fast feature pyramid theory that introduced in Section 4.1.2, parameter λ is calculated by employing the extracted feature data during the training data. Therefore, the parameter λ that calculated by using pedestrian detection dataset is incorrect for the traffic sign detection application. From the feature scaler IP, we know that λ is a parameter employed during the approximate computing. Hence, the configuration of each feature scaler IP should be modified.

- Secondly, due to the difference of the size of sliding window, the number of scaled images is changed. In the pedestrian detection system, the size of sliding window is 128×64. According to the fast feature pyramid theory, 19 scales are employed from 640×480 to 128×64. However, in the traffic sign detection application, the size of the sliding window is 56×56. Therefore, more scales are used to detect the traffic signs from $640\times$ to 56×56. Through analysis, we found that it is meaningless to detect the traffic signs that are too big. Considering the BRAM consumption in the design of pedestrian detection system, more feature scaler can not be added to the system. Therefore, the range of the size of traffic signs that can be detected by the system should be defined.

- Besides, the detector that obtained in the training phase is also different for different applications. Hence, the information inside the detector, such as the confidence value of each decision tree and tree index, which is utilized during the classification process, should also be updated.

After these modifications based on a replicate single camera based pedestrian detection system, the traffic sign detection system is created. According to the validation experiments of the traffic sign detection application that introduced in Section 4.3, two versions of detection system are created: one detects each category of traffic signs separately and the other employs the detector to detect all traffic signs together. Therefore, real-time (24 fps) traffic sign detection systems are achieved on the SoC-FPGA platform with minimum effort. Some examples of

the detected results are given in Figure 6.9.

Figure 6.9: Illustrations of detection results of the traffic sign detection system

Due to that the system is running on the Zedboard with live camera, it is difficult to measure the miss rate or performance of the system. After multiple times of test with different traffic signs under various angles, the testing results show that the system is more robust for prohibitory and danger traffic signs, which is consistent with the results of framework validation by traffic sign detection application.

The resource utilization report of the traffic sign detection system is listed in Table 6.3. Although two versions of traffic sign detection system are implemented, since the only difference between two systems is the classification module, same hardware design is employed. Therefore, only one resource utilization report is given in Table 6.3. Besides, the resource utilization report of each module inside the system is similar to the report of the pedestrian detection system, the detailed resource utilization report is omitted here. By comparing the resource utilization reports of the pedestrian detection system and the traffic sign detection system, we can obtain that slightly less BRAMs are employed in the traffic sign detection system, which is caused by the configuration difference of the feature scaler IP cores.

Type	Resources Used	Available
LUT	33679	53,200
FF	43926	106,400
BRAM	117.5	147
DSP	156	220

Table 6.3: The resource utilization report of traffic sign detection system on Zedboard

6.4 Design of Head Detection System on a SoC-FPGA Platform

In this section, a head detection system is designed and tested on the object detection platform. Similar to the traffic sign detection system that designed in Section 6.3, the configurable parameters of all feature extraction IPs are revised according to the fast feature pyramid theory. Moreover, the detector that obtained in the training phase of the traffic sign detection algorithm is also updated.

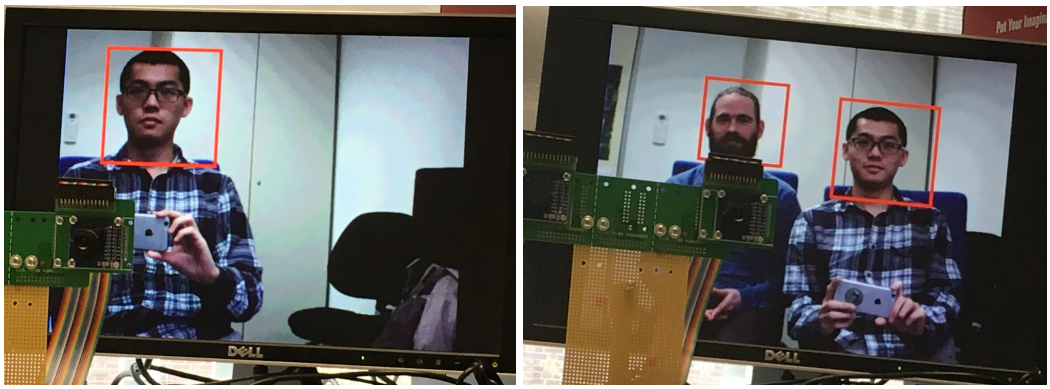


Figure 6.10: Illustrations of detection results of the head detection system

Figure 6.10 illustrates the detection results of the designed head detection system. In this application, only the heads in front view are trained by the head detection algorithm. Therefore, the system can only detect the head of the humans from the front view. The resource utilization report of the head detection system is listed in Table 6.4. Since the size of the sliding window that employed in both the head detection system and the traffic sign detection system is the same, which equals 56×56, similar amount of resources are used by the head detection system, compared with the traffic sign detection system.

Type	Resources Used	Available
LUT	33754	53,200
FF	43961	106,400
BRAM	117.5	147
DSP	140	220

Table 6.4: The resource utilization report of head detection system on Zedboard

In this thesis, only pedestrian detection system, head detection system, and traffic sign detection system are designed and tested. In fact, with the generalized object

detection framework and the designed IP-cores, the object detection system that targets any instance of objects (e.g., face) can be designed and implemented rapidly.

6.5 Summary

In this chapter, a video-based object detection platform, which consists of two OV7670 cameras, Zedboard, and a monitor, is created. Based on this platform and the IP cores that designed in Chapter 5, different object detection systems are designed and tested. The detection performance and result utilization report of each designed system are given.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, different object detection systems (e.g., pedestrian detection system, traffic sign detection system) are designed and evaluated on a SoC-FPGA platform. The main contributions of this work include five parts:

- Firstly, representative object detection algorithms are selected, implemented, and evaluated.
- Secondly, a generalized object detection framework is created. Based on this framework, pedestrian detection, traffic sign detection, and head detection algorithms are realized and tested.
- Thirdly, a video-based object detection platform is created, which includes stereo cameras, Zedboard, and a monitor. To achieve camera and Zedboard connection via Pmod interface, two PCB boards are designed.
- Fourthly, an IP-toolbox which contains all of the IP cores that correspond to each block of the generalized object detection framework is designed.
- Finally, real-time pedestrian detection system, traffic sign detection system, and head detection system are designed and realized on the Zedboard.

More details of each part are summarized as follows.

Algorithm performance and execution speed are two typically employed factors to evaluate an object detection system. Hence, promising algorithm obtaining is crucial. To achieve this, many representative and state-of-art object detection algorithms, especially pedestrian detection algorithms, are selected, implemented, and evaluated. During this process, comparisons are made to analyze the impact of different features, training methods, and training configurations. The visualizations of different object detectors are generated to analyze the performance dif-

ference between different detectors. Among all of the implemented algorithms, DeepPed [8] achieves the best performance compared with other realized algorithms. However, DeepPed is based on the convolutional neural networks, which increases the difficulty of designing different object detection systems based on the same group of IP cores. Finally, the ACF algorithm [34], which achieves the second best accuracy during the evaluation, is selected as the prototype algorithm for hardware design.

Based on the fast feature pyramid theory proposed by FPDW [48] and ACF [34], a generalized object detection framework is created. By employing this framework with different configurations and datasets, different object detection applications can be realized. In this thesis, pedestrian detection, traffic sign detection, and head detection are implemented by adapting the framework with different configurations. The validation results reveal that promising performance can be achieved by employing this framework to different object detection applications.

When the algorithm is prepared, a video-based object detection platform is thereafter created. In order to get the depth information of detected objects, stereo cameras are employed and soldered on a stable support. Then, stereo camera calibration and rectification are processed by employing a checkerboard.

After the object detection platform preparation, IP cores that correspond to each block of the generalized object detection framework are designed and implemented. To make the IP cores handle different scenarios, configurable parameters are provided by feature calculation IP, feature scaler IP, feature data merger IP, drawing IP, etc.

Finally, by customizing and connecting different IPs according to the fast feature pyramid theory, different object detection systems are designed, implemented, and tested. Due to the limitation of resources on Zedboard, the feature extraction is performed on selected image scales whose scaling parameters are calculated by Matlab. The hardware synthesis reports show that the resources on Zedboard permit feature extraction from maximum ten scaled images. Therefore, pedestrian detection system, traffic sign detection system and head detection system that utilizes ten scaled images' data are realized. From on board testing, we can conclude that real-time object detection is achieved by the designed detection systems. Besides, with this generalized object detection framework and the designed IP-cores, any other object detection systems (e.g., car detection, face detection) can also be designed and implemented on a SoC-FPGA platform rapidly.

7.2 Future Work

The object detection systems that designed and implemented in this work are all based on two OV7670 cameras whose resolution is 640×480. In order to achieve object detection with better performance, high-resolution cameras, such as HD camera OV2640 [87] whose interface is the same as OV7670, can be employed as the data input of the system.

The targeting platform of this work is a Xilinx Zedboard, which is a widely used SoC-FPGA with limited resources on board. Due to its resource limitation, the feature extraction module only extracts the feature data under ten different image scales. Therefore, employing a SoC-FPGA with more resources (programming logic, BRAM, DSP, etc.) could also improve the accuracy of the designed object detection system.

Bibliography

[1] Y. Amit and P. Felzenszwalb, *Object Detection*, pp. 537–542. Boston, MA: Springer US, 2014.

[2] "Waymo: The First Self-Driving Car From Google." `https://waymo.com/tech`.

[3] W. Wachenfeld, H. Winner, J. C. Gerdes, B. Lenz, M. Maurer, S. Beiker, E. Fraedrich, and T. Winkle, *Use Cases for Autonomous Driving*, pp. 9–37. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[4] A. Abdulkhaleq, D. Lammering, S. Wagner, J. Röder, N. Balbierer, L. Ramsauer, T. Raste, and H. Boehmert, "A Systematic Approach Based on STPA for Developing a Dependable Architecture for Fully Automated Driving Vehicles," *Procedia Engineering*, vol. 179, pp. 41 – 51, 2017.

[5] P. Jiang, J. Peng, G. Zhang, E. Cheng, V. Megalooikonomou, and H. Ling, "Learning-Based Automatic Breast Tumor Detection and Segmentation in Ultrasound Images," in *9th IEEE International Symposium on Biomedical Imaging (ISBI)*, pp. 1587–1590, May 2012.

[6] S. Zhang, R. Benenson, and B. Schiele, "Filtered Channel Features for Pedestrian Detection," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1751–1760, June 2015.

[7] S. Zhang, R. Benenson, M. Omran, J. Hosang, and B. Schiele, "How Far are We From Solving Pedestrian Detection?," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1259–1267, June 2016.

[8] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro, "Deep Convolutional Neural Networks for Pedestrian Detection," *Signal Processing: Image Communication*, vol. 47, pp. 482 – 489, Sep. 2016.

[9] I. Orozco, M. E. Beumi, and J. J. Berlles, "A Study on Pedestrian Detection Using a Deep Convolutional Neural Network," in *International Conference on Pattern Recognition Systems (ICPRS-16)*, pp. 1–4, April 2016.

[10] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-Based

Real-Time Pedestrian Detection on High-Resolution Images," in *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 629–635, June 2013.

[11] H. Xiao, H. Song, W. He, and K. Yuan, "Real-Time Shape and Pedestrian Detection With FPGA," in *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 2381–2386, Aug 2015.

[12] A. Khan, M. U. K. Khan, M. Bilal, and C. M. Kyung, "Hardware Architecture and Optimization of Sliding Window Based Pedestrian Detection on FPGA for High Resolution Images by Varying Local Features," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 142–148, Oct 2015.

[13] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "Fast FPGA-Based Architecture for Pedestrian Detection Based on Covariance Matrices," in *18th IEEE International Conference on Image Processing*, pp. 389–392, Sep. 2011.

[14] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, June 2005.

[15] "Xilinx Vivado Design Suite." `products/design-tools/vivado.html`.

[16] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119 – 139, 1997.

[18] S. Zhou, H. Greenspan, and D. Shen, *Deep Learning for Medical Image Analysis*. Elsevier Science, 2017.

[19] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral Channel Features," 2009.

[20] F. Billmeyer and M. Saltzman, *Principles of Color Technology*. Wiley-Interscience Publication, Wiley, 1981.

[21] "The CIE Website." `http://www.cie.co.at/cie`.

[22] "The INRIA Person Dataset." `http://pascal.inrialpes.fr/data/human`.

[23] R. Gonzalez and R. Woods, *Digital Image Processing*. Pearson/Prentice Hall, 2008.

[24] P. Viola and M. Jones, "Robust Real-Time Object Detection," *International Journal of Computer Vision*, vol. 4, 2001.

[25] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[26] R. Shah, *Support Vector Machines for Classification and Regression.* McGill theses, 2007.

[27] P. Vasuki and S. Veluchamy, "Pedestrian Detection for Driver Assistance Systems," in *2016 International Conference on Recent Trends in Information Technology (ICRTIT)*, pp. 1–4, April 2016.

[28] W. X. Li, W. L. Ma, B. Quan, M. x. Pei, and X. x. Feng, "A Pedestrian Detection Method Based on PSO and Multimodal Function," in *2016 Chinese Control and Decision Conference (CCDC)*, pp. 6054–6058, May 2016.

[29] J. Song, T. Wu, and P. An, "Cascade Linear SVM for Object Detection," in *The 9th International Conference for Young Computer Scientists*, pp. 1755–1759, Nov 2008.

[30] Z. Duan, L. Miao, and H. Wang, "Binarized Normed Gradients for Object Detection," in *2015 IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 1064–1068, Dec 2015.

[31] J. Tang, J. Li, and X. An, "Learning Proposal for Front-Vehicle Detection," in *2015 Chinese Automation Congress (CAC)*, pp. 773–777, Nov 2015.

[32] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining: Pearson New International Edition.* Pearson Education Limited, 2013.

[33] R. Schapire and Y. Freund, *Boosting: Foundations and Algorithms.* MIT Press, 2012.

[34] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast Feature Pyramids for Object Detection," *PAMI*, 2014.

[35] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool, "Pedestrian Detection at 100 Frames per Second," in *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2903–2910, IEEE, 2012.

[36] L. Lu, Y. Zheng, G. Carneiro, and L. Yang, *Deep Learning and Convolutional Neural Networks for Medical Image Computing: Precision Medicine, High Performance and Large-Scale Datasets.* Advances in Computer Vision and Pattern Recogni-

tion, Springer International Publishing, 2017.

[37] H. Aghdam and E. Heravi, *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer International Publishing, 2017.

[38] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]," *IEEE Computational Intelligence Magazine*, vol. 5, pp. 13–18, Nov 2010.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[40] J. Heaton, *Artificial Intelligence for Humans: Nature-inspired Algorithms*. Artificial Intelligence for Humans, Heaton Research, Incorporated, 2014.

[41] "Online Course: Convolutional Neural Networks for Visual Recognition." `http://cs231n.github.io`, March 2017.

[42] "The Caltech Pedestrian Dataset." `http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/datasets/USA`.

[43] I. M. Creusen, R. G. Wijnhoven, E. Herbschleb, and P. de With, "Color Exploitation in Hog-Based Traffic Sign Detection," in *17th IEEE International Conference on Image Processing (ICIP)*, pp. 2669–2672, IEEE, 2010.

[44] Y. Xie, L.-f. Liu, C.-h. Li, and Y.-y. Qu, "Unifying Visual Saliency With HOG Feature Learning for Traffic Sign Detection," in *Intelligent Vehicles Symposium*, pp. 24–29, IEEE, 2009.

[45] T. Li, X. Cao, and Y. Xu, "An Effective Crossing Cyclist Detection on a Moving Vehicle," in *8th World Congress on Intelligent Control and Automation (WCICA)*, pp. 368–372, 2010.

[46] "The Open Source Computer Vision Library." `http://opencv.org`.

[47] "The Robot Operating System Detection Library." `http://wiki.ros.org/find_object_2d`.

[48] P. Dollár, S. J. Belongie, and P. Perona, "The Fastest Pedestrian Detector in the West," in *BMVC*, vol. 2, p. 7, Citeseer, 2010.

[49] B. Yang, J. Yan, Z. Lei, and S. Z. Li, "Convolutional Channel Features," in *Proceedings of the IEEE international conference on computer vision*, pp. 82–90, 2015.

[50] Y. Tian, P. Luo, X. Wang, and X. Tang, "Pedestrian Detection Aided by Deep Learning Semantic Tasks," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5079–5087, June 2015.

[51] J. Hosang, M. Omran, R. Benenson, and B. Schiele, "Taking a Deeper Look at Pedestrians," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4073–4082, 2015.

[52] D. Ribeiro, A. Mateus, J. C. Nascimento, and P. Miraldo, "A Real-Time Pedestrian Detector using Deep Learning for Human-Aware Navigation," 2016.

[53] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A Convolutional Neural Network Cascade for Face Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5325–5334, 2015.

[54] Y. Wei, X. Bing, and C. Chareonsak, "FPGA Implementation of AdaBoost Algorithm for Detection of Face Biometrics," in *2004 IEEE International Workshop on Biomedical Circuits and Systems*, pp. S1–6, IEEE, 2004.

[55] A. Suleiman and V. Sze, "Energy-Efficient HOG-Based Object Detection at 1080HD 60 fps With Multi-Scale Support," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, IEEE, 2014.

[56] C. Kyrkou and T. Theocharides, "A Flexible Parallel Hardware Architecture for AdaBoost-Based Real-Time Object Detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 1034–1047, 2011.

[57] W. Shi, X. Li, Z. Yu, and G. Overett, "An FPGA-Based Hardware Accelerator for Traffic Sign Detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1362–1372, 2017.

[58] P. Janus, K. Piszczek, and T. Kryjak, *FPGA Implementation of the Flux Tensor Moving Object Detection Method*, pp. 486–497. Cham: Springer International Publishing, 2016.

[59] "YOLO: Real-Time Object Detection." `https://pjreddie.com/darknet/yolo`.

[60] P. Dollár, "Piotr's Computer Vision Matlab Toolbox (PMT)." `https://github.com/pdollar/toolbox`.

[61] "LIBSVM – A Library for Support Vector Machines." `https://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[62] "The Source Code of the VeryFast Algorithm." http://rodrigob.github.io/#contact.

[63] "CUDA C Programming Guide." http://docs.nvidia.com/cuda/cuda-c-programming-guide, March 2017.

[64] "Caffe: A Deep Learning Framework." http://caffe.berkeleyvision.org.

[65] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Computer Vision and Pattern Recognition*, 2014.

[66] "The Technical Specification of Zedboard." http://zedboard.org/sites/default/files/product_briefs/PB-AES-Z7EV-7Z020-G-V2.pdf.

[67] D. L. Ruderman and W. Bialek, "Statistics of Natural Images: Scaling in the Woods," *Phys. Rev. Lett.*, vol. 73, pp. 814–817, Aug 1994.

[68] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks*, no. 1288, 2013.

[69] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition," *Neural Networks*, no. 0, pp. –, 2012.

[70] "The Head Detection Dataset." http://www.vision.caltech.edu/html-files/archive.html.

[71] S. Yang, P. Luo, C.-C. Loy, and X. Tang, "Wider Face: A Face Detection Benchmark," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5525–5533, 2016.

[72] "The OV7670 Camera." http://www.arducam.com/camera-modules/0-3mp-ov7670.

[73] "Hamster's Work of OV7670 on Zedboard." http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_OV7670.

[74] "The OmniVision Technologies." http://www.ovt.com.

[75] OmniVision, *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*, version 2.2 ed., June 2007.

[76] "The Colorchecker Pages." http://www.babelcolor.com/colorchecker.

htm.

[77] "Colorchecker Specification Document." http://xritephoto.com/ph_product_overview.aspx?ID=938&Action=Support&SupportID=5884.

[78] G. D. Finlayson, M. Mackiewicz, and A. Hurlbert, "Color Correction Using Root-Polynomial Regression," *IEEE Transactions on Image Processing*, vol. 24, pp. 1460–1470, May 2015.

[79] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[80] Z. Zhang, "Flexible Camera Calibration by Viewing a Plane From Unknown Orientations," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pp. 666–673 Vol.1, 1999.

[81] "Camera Calibration Toolbox for Matlab." http://www.vision.caltech.edu/bouguetj/calib_doc, 2015.

[82] S. Qin and M. Berekovic, "A Comparison of High-Level Design Tools for SoC-FPGA on Disparity Map Calculation Example," *CoRR*, vol. abs/1509.00036, 2015.

[83] "Divider Generator Version 5.1." https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf.

[84] "CORDIC Intellectual Property." https://www.xilinx.com/products/intellectual-property/cordic.html.

[85] "VGA Signal Timing Industry Standard." http://tinyvga.com/vga-timing.

[86] "RGB to YUV Color Space Conversion." http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_VGA_HDMI.

[87] "The OV2640 Camera." http://www.arducam.com/camera-modules/2mp-ov2640.